A COMPREHENSIVE STUDY OF DEEP BIDIRECTIONAL LSTM RNNS FOR ACOUSTIC MODELING IN SPEECH RECOGNITION

Albert Zeyer, Patrick Doetsch, Paul Voigtlaender, Ralf Schlüter, Hermann Ney

Human Language Technology and Pattern Recognition, Computer Science Department,
RWTH Aachen University, 52062 Aachen, Germany
{zeyer, doetsch, voigtlaender, schlueter, ney}@cs.rwth-aachen.de

ABSTRACT

Recent experiments show that deep bidirectional long short-term memory (BLSTM) recurrent neural network acoustic models outperform feedforward neural networks for automatic speech recognition (ASR). However, their training requires a lot of tuning and experience. In this work, we provide a comprehensive overview over various BLSTM training aspects and their interplay within ASR, which has been missing so far in the literature. We investigate on different variants of optimization methods, batching, truncated backpropagation, and regularization techniques such as dropout, and we study the effect of size and depth, training models of up to 10 layers. This includes a comparison of computation times vs. recognition performance. Furthermore, we introduce a pretraining scheme for LSTMs with layer-wise construction of the network showing good improvements especially for deep networks. The experimental analysis mainly was performed on the Quaero task, with additional results on Switchboard. The best BLSTM model gave a relative improvement in word error rate of over 15% compared to our best feed-forward baseline on our Quaero 50h task. All experiments were done using RETURNN and RASR, RWTH's extensible training framework for universal recurrent neural networks and ASR toolkit. The training configuration files are publicly available.

Index Terms— acoustic modeling, LSTM, RNN

1. INTRODUCTION AND RELATED WORK

Deep neural networks (DNN) yield state-of-the-art performance in classification in many machine learning tasks [1]. The class of recurrent neural networks (RNN) and especially long short-term memory (LSTM) networks [2] perform very well when dealing with sequence data like speech.

Only recently, it has been shown that LSTM based acoustic models (AM) outperform FFNNs on large vocabulary continuous speech recognition (LVCSR) [3, 4]. The training procedure for LSTMs, esp. deep bidirectional LSTMs (BLSTM) takes a lot of time and effort to tune, arguably more than for feed-forward networks. There are many aspects to be considered for training LSTMs which we are exploring in this work, such as the network topology, sequence chunking and batch sizes, optimization methods, regularization, and our experiments show that there is a huge variance in recognition performance depending on all the different aspects. What is missing, is an overview over the effect and interdependencies of the various approaches. To the best of our knowledge, currently no overview like this exists the literature, and this is presented in this work. We try to fill this gap by a comprehensive study of various aspects of training deep BLSTMs and we provide configuration files for all our experiments [5] for our framework RETURNN [6]. Compared to our best FFNN baseline, we get a relative improvement in word error rate (WER) of over 15%. We train deep BLSTM networks with up to 10 layers for acoustic modeling and we discovered that a pretraining scheme with a layer-wise construction can improve performance for deeper LSTMs. We are not aware of any previous work which applied pretraining for LSTMs in ASR.

Hybrid RNN-HMM models were developed in 1994 in [7]. An early work for bidirectional RNNs for TIMIT was presented in [8] and an early hybrid LSTM-HMM was presented in [9] for TIMIT. [3, 4, 10, 11, 12, 13, 14, 15] investigate various bidirectional and unidirectional LSTM topologies with optional projection in some cases combined with convolutional or feed-forward layers for acoustic modeling in ASR. Variations of the LSTM model were studied in [16, 17, 18, 19], although we only present the standard LSTM without peephole in this work.

2. LSTM MODEL AND IMPLEMENTATION

We use the standard LSTM model without peephole connections [20]. If not otherwise stated, we use bidirectional LSTMs (BLSTM). Our base tool is the RASR speech recognition toolkit [21, 22]. We use RASR for the feature extraction pipeline and for decoding. We extended RASR with a Python bridge to allow many kinds of interactions with external tools. This Python bridge was introduced to be able to use RETURNN, our Theano-based framework [6, 23] to do the training and forwarding in recognition of our acoustic model. In RETURNN, we have multiple LSTM implementations and it supports all the aspects which we discuss in this paper. One particular LSTM implementation is supported by a custom CUDA kernel which gives us great speed improvements. We provide more details about this software in [6] and the config files in [5].

3. COMPARISONS AND EXPERIMENTS

We use a subset of 50 hours from the Quaero Broadcast Conversational English Speech database *train11*. The development *eval10* and evaluation *eval11* sets consist of about 3.5 hours of speech each. The recognition is performed using a 4-gram language model. Further details about the task can be found in [24].

3.1. Baseline

We use the common NN-HMM hybrid acoustic model [25]. All acoustic models were trained frame-wise with the cross entropy criterion based on a fixed Viterbi alignment. We do not investigate discriminative sequence training in this study. The input features are 50-dimensional VTLN-normalized Gammatone [26]. We don't add any context window nor delta frames for the LSTM because we expect that the LSTM automatically learns to use the context. We use a Classification And Regression Tree (CART) with 4501 labels. We also have special residual phoneme types in our lexicon which are used in transcription for unknown or unintelligible parts. We remove all frames which are aligned to such phonemes according to our fixed Viterbi alignment. This means that we have only 4498 output class labels in our softmax layer and in recognition, we never hypothesize such phonemes.

Our FFNN baseline with 9x2000 layers and ReLU activation function with a context-window of 17 frames yields 15.3% WER on *eval10* and 20.3% WER on *eval11*.

Our minibatch construction is similar to e. g. [10] and described in detail in [6]. One minibatch consists of n_{chunks} number of chunks from one or more corpus segments. The chunks are up to T frames long and we select them every t_{step} frames from the corpus. Our common settings are $T = 50$, $t_{\text{step}} = 25$, $n_{\text{chunks}} = 40$, i. e. a minibatch size of 2000 frames.

Our learning rate is not normalized by $\sum_i T_i$ or $T \cdot n_{\text{chunks}}$ so that the update step stays the same for every mini batch independent from n_{chunks} or T. Thus, in our case, the total update scale per epoch stays the same independent from n_{chunks} or T. Only t_{step} will have an impact on the total update scale.

For all experiments, we train 30 epochs. We have a small separate cross validation (CV) set where we measure the frame error rate (FER) and the cross entropy (CE). With the model from epochs 5, 10, 30, the epoch from the best CV FER, the epoch from the best CV CE, we evaluate on *eval10* and *eval11*. In the results in our tables, we select the epoch of the best WER on *eval10*. We also state the epoch. This can give a hint about the convergence speed or whether we overfit later.

Despite the optimization method which might already provide some kind of implicit learning rate scheduling, we always also use another explicit learning rate scheduling method which is often called Newbob [6]. We start with some given initial learning rate and when the relative improvement on the CV CE is less than 0.01 after an epoch, we multiply the learning rate with 0.5 for the next epoch.

Our standard optimization method is most often Adam [27] with an initial learning rate of 10^{-3}. We use gradient clipping of 10 by default.

3.2. Number of Layers

We did several experiments to figure out the optimal number of layers. In theory, more layers should not hurt but in practice, they often do because the optimization problem becomes harder. This could be overcome with clever initializations, skip connections, highway network like structures [28, 15] or deep residual learning [29]. We did some initial experiments also in that direction but we were not successful so far. The existing work in that direction is also mostly for deep FFNNs and not for deep RNNs except for [15] which trains deep highway BLSTMs up to 8 layers.

The results can be seen in Table 1. For this experiment, the optimum is somewhere between 4 to 6 layers. In earlier experiments, the optimum was at about 3 to 4 layers. It seems the more we improve other hyperparameters, the deeper the optimal network becomes. With pretraining as in Section 3.8, we get our overall best result with 6 layers.

We also included the best CE value on the train dataset and the CV dataset in Table 1. This gives a hint about the amount of overfitting. We observe similar results as in [29], i. e. deeper networks should in theory overfit even more but they do not which is probably due to a harder optimization problem. It also seems as if the CV CE optimum is slightly deeper than the WER optimum. That indicates that sequence discriminative training will further improve the results.

Table 1: Comparison of number of layers, layer size fixed to 500 for each forward and backward direction. Dropout 0.1 + L_2, Adam, $n_{\text{chunks}} = 40$, WER on *eval10*, reported on the best epoch. Note that the CE values are not necessarily from the same epoch as the WER but they are the minimum from all epochs. Also, the train CE is accumulated while training, i.e. with dropout applied.

#layers	#params[M]	WER[%]	epoch	train CE	CV CE
1	6.7	17.6	30	1.72	1.64
2	12.7	14.6	16	1.25	1.39
3	18.7	14.0	30	1.17	1.32
4	24.7	**13.5**	15	**1.16**	1.29
5	30.7	13.6	30	1.17	**1.28**
6	36.7	**13.5**	30	1.22	**1.28**
7	42.7	13.8	30	1.24	**1.28**
8	48.7	14.2	19	1.29	1.31

3.3. Layer Size

In most experiments, we use a hidden layer size of 500 (i. e. 500 nodes / memory cells for each the forward and the backward direction). In Table 2 we compare different layer sizes. Note that the number of parameters increases quadratically. We see that the optimum for this experiment is at about 600-700 (for 3 layers with Adadelta at about 700), however a model with size 500 is much smaller and not so much worse, so we used that size for most other experiments.

We did not investigate projections in this work. With a projection size of about 500, other groups report a layer size of up to 2000 [3].

Table 2: Comparison of hidden layer size. 5 layers, dropout 0.1, L_2 0.01, Adam, $n_{\text{chunks}} = 40$. WER reported on *eval10*, reported for the best epoch.

layer size	#params[M]	WER[%]	epoch
500	30.7	13.6	30
600	43.1	**13.5**	30
700	57.6	**13.5**	18
800	74.1	13.6	30

3.4. Topology: Bidirectional vs. Unidirectional

Our original experiment showed that we get quite a huge WER degradation with unidirectional LSTM networks compared to BLSTMs, over 20% relative, 19.6% WER for unidirectional vs. 15.6% WER for bidirectional, see [30], although we did not tune the unidirectional network as much. Other groups confirm that bidirectional networks perform better than unidirectional ones [8, 31].

This huge WER degradation led to further research where we investigated how to use bidirectional RNNs/LSTMs on a continuous input sequence to do online recognition. We showed that this is possible and with some recognition delay, we can reach the original WER. These results are described in [30].

3.5. Batching

We investigated the effect of different numbers of chunks n_{chunks}, window time steps t_{step} and window maximum size T, resulting in the overall batch size $T \cdot n_{\text{chunks}}$. All experiment were done with the same initial learning rate. We did many experiments with varying $n_{\text{chunks}} \in \{20, \ldots, 80\}$ and got the best results with $n_{\text{chunks}} \approx 40$. For some experiments the performance difference was quite notable better with $n_{\text{chunks}} = 40$ compared to $n_{\text{chunks}} = 20$. This might be because of a better variance and thus more stable gradient for each minibatch. Note that a higher n_{chunks} is usually also faster up to a certain point because the GPU can work in parallel on every chunk. We usually use $T = 50$. We did many experiments with fixed $T - t_{\text{step}} = 25$ but we often see a slight degradation when $T \geq 100$. This might be due to the problem being harder to train because of the longer backpropagation through time but maybe we need to tune the learning rate or other parameters more for longer chunks. Varying t_{step} did not make much difference except that for smaller t_{step}, the training time per epoch naturally becomes longer because we see some of the data more often.

3.6. Optimization Methods

We compare many optimization methods and variations between hyperparameters and esp. also different initial learning rates in Table 3. We compare stochastic gradient descent (SGD), SGD with momentum [32, 33] where one variant only depends on the last minibatch (*mom*) and another variant depends on the full history (*mom2*), SGD with Nesterov momentum [34, 33], mean-normalized SGD (MNSGD) [35], Adadelta [36], Adagrad [37], Adam and Adamax [27], Adam without the learning rate decay term, Nadam (Adam with incorporated Nesterov momentum) [38], Adam with gradient noise [39], Adam with MNSGD combined, RMSprop [40] and an RMSprop inspired method called SMORMS3 [41]. We also tried Adasecant [42] but it did not converge in any of our experiments for this ASR task. We also test the effect of Newbob. Note that we only use 3 layers, no L_2 and a smaller n_{chunks}, which leads to worse results here compared to some other sections.

One notable variant was also to use several model copies n which we update independently and which we merge together by averaging after some k minibatch updates (*upd-mm-n-k*). We vary the amount of model copies and after how much batches we merge. This is similar to the multi-GPU training behavior described in [6]. This method yielded the best result in these experiments but we postpone this for further research.

Overall, Adam was always a good choice. Standard SGD comes close in some experiments but converges slower. Newbob was also important. Note that Newbob also has some hyperparameters and tuning those will likely yield further improvements.

We also investigated the effect of various different gradient clipping variants and we settled with clipping the total gradient for all parameters with a value of 10, which stabilized the training in some cases, although if possible, no clipping yields the best performance in many cases.

Table 3: Comparison of different optimization methods. 3 layers, hidden layer size 500, dropout 0.1, $n_{\text{chunks}} = 20$. WER5, WER10, bWER and ep is the *eval10* WER[%] of epoch 5, 10, best WER[%] and the epoch of the best WER, respectively.

method	lr	details	WER5	WER10	bWER	ep
SGD	10^{-3}	-	17.0	16.1	15.8	30
	10^{-4}	-	17.9	15.8	14.9	26
		mom 0.9	17.4	15.9	14.8	28
		mom2 0.9	16.7	16.3	15.9	19
		mom2 0.5	17.2	16.0	15.0	30
		Nesterov 0.9	16.9	16.1	15.8	16
	$0.5 \cdot 10^{-4}$	-	19.7	17.1	15.4	30
		mom2 0.9	16.8	15.5	15.0	30
	10^{-5}	-	32.1	22.3	18.6	30
		then lr 10^{-4}	18.7	16.2	15.0	30
MNSGD	10^{-4}	avg 0.5	20.2	18.2	17.8	20
		avg 0.995	19.1	16.8	16.4	18
RMSprop	10^{-3}	mom 0.9	33.7	26.5	26.5	10
SMORMS3	10^{-3}	-	16.4	16.0	15.7	23
	10^{-3}	mom 0.9	16.8	16.5	15.6	29
Adadelta	0.5	decay 0.90	20.2	15.7	15.3	13
		decay 0.95	18.4	15.7	15.1	13
		decay 0.99	model broken			
	0.1	decay 0.95	16.9	15.5	15.1	13
	10^{-2}	decay 0.95	24.4	20.1	17.4	29
Adagrad	10^{-2}	-	16.9	16.0	15.6	29
	10^{-3}	-	model broken			
Adam	10^{-2}	-	model broken			
	10^{-3}	-	16.3	15.4	14.8	30
		no lr decay	16.1	15.0	14.6	11
		Nadam	16.1	14.8	14.7	30
		grad noise 0.3	16.2	15.0	14.6	16
		upd-mm-2-2	15.8	14.9	14.5	18
		upd-mm-3-2	15.8	14.5	**14.3**	30
		Adamax	16.3	15.4	14.9	15
	$0.5 \cdot 10^{-3}$	-	15.8	14.9	14.5	13
	10^{-4}	-	16.4	15.6	14.9	18
		Adamax	21.0	18.6	16.6	30
		MNSGD	16.5	15.7	14.9	18
		no Newbob	16.4	15.6	15.2	21
	10^{-5}	no Newbob	30.7	24.3	19.2	30

3.7. Regularization Methods

For regularization, we tried both dropout [43] and standard L_2 regularization. The optimal dropout factor depends on the hidden size and many other aspects, although we mostly see the optimal WER with dropout 0.1, i. e. we drop 10% of the activations and multiply by $\frac{10}{9}$. If we enlarge the hidden layer size, we can use higher dropout values although in most experiments, dropout 0.2 was worse than dropout 0.1.

Interestingly, the combination of both L_2 and dropout gives a big improvement and yields the best result. See Table 4.

3.8. Initialization and Pretraining

In all cases, we randomly initialize the parameters similar to [44].

We investigated the same pretraining scheme as we do for our FFNN where we start with one layer and add a layer after each epoch right before the output layer [45]. In each pretrain epoch, we can either train only the new layer (greedily) or the full network, where full network training usually was better.

Results can be seen in Table 5. For deeper networks, this

Table 4: We try different combinations of dropout and L_2. 3 layers, hidden size 500, $n_{\text{chunks}} = 40$, Adam. WER on *eval10*.

dropout	L_2	WER[%]	epoch
0	0	16.1	6
	10^{-2}	14.8	11
0.1	0	14.8	19
	10^{-3}	14.5	11
	10^{-2}	**14.0**	30
	10^{-1}	15.2	26

scheme seems to help more. This indicates that our initialization might have room for improvement. We got our overall best result with such a pretraining scheme applied for a 6 layer bidirectional LSTM and we note that esp. for the deeper networks, the improvements by pretraining increases. We were not able to train a 9 layer BLSTM without pretraining, it diverged and broke after two epochs. Also, the training calculation time of the first few epochs is shorter.

Table 5: Comparison of pretraining and no pretraining, compared for different final number of layers, cf. Table 1.

#layers	WER[%]	
	no pretrain	pretrain
1	17.6	-
2	14.6	14.4
3	14.0	13.7
4	**13.5**	13.4
5	13.6	13.5
6	**13.5**	**13.0**
7	13.8	13.1
8	14.2	13.3
9	broken	13.3
10	-	13.3

3.9. Calculation Time vs. WER

We did over 300 different training experiments and collected a lot of statistics about the calculation time in relation to the WER.

Most experiments were done with a GeForce GTX 980. We see that the Tesla K20c is about 1.38 times slower with a standard deviation of 0.084, and the GeForce GTX 680 is about 1.86 times slower with a standard deviation of 0.764. We present the pure train epoch calculation times with a GeForce GTX 980, not counting the CV test and other epoch preparation.

We collected some of the total times in Table 6. That is the summed train epoch time until we reach the specific epoch. We show the model with the best WER up to the specific time. We see that in most cases, combinations of different hyperparameters and methods yield the best results. Time downsampling was a simple method to reduce the calculation time with performance as trade-off.

4. EXPERIMENTS ON OTHER CORPORA

We use the 300h Switchboard-1 Release 2 (LDC97S62) corpus for training and the Hub5'00 evaluation data (LDC2002S09) is used for testing. We use a 4-gram language model which was trained on the transcripts of the acoustic training data (3M running words) and the transcripts of the Fisher English corpora (LDC2004T19 & LDC2005T19) with 22M running words. More details can be found in [46]. Results in Table 7 show the improvement by LSTMs and also with an associative LSTM [47].

Table 6: Total times until we get to a certain *eval10* WER in a certain train epoch. If not specified, it's a BLSTM.

time	WER	ep	model	
	[%]		size	details
2:18h	20.0	5	1x500	dropout + L_2
2:36h	17.2	5	3x300	dropout + time downsampling
3:40h	16.6	5	3x500	dropout
12:30h	15.3	20	9x2000	FFNN with relu
14:47h	13.9	13	3x500	dropout + L_2
21:53h	13.6	17	4x500	dropout + L_2 + pretraining
35:36h	13.2	18	5x500	dropout + L_2 + grad noise
41:51h	13.0	23	6x500	dropout + L_2 + pretraining

Table 7: Results on Switchboard. BLSTM models trained with Nadam, gradient noise, dropout + L_2. Additionally with an associative LSTM layer on top. Cf. Section 4.

model	WER[%]		
	total	SWB	CH
FFNN	19.4	13.1	25.6
5l. BLSTM	17.1	11.9	22.3
6l. BLSTM	16.7	11.5	21.9
5l. BLSTM + assoc. BLSTM	**16.3**	**11.1**	**21.6**

We also did a few experiments on Babel Javanese full language pack (`IARPA-babel402b-v1.0b`) which is a keyword-search (KWS) task (see [48] for all details). The baseline FFNN with 6 layers and 34M parameters yields a WER of 54.3% with CE-training and 53.3% with MPE-training. A 3 layer BLSTM with 19M parameters yields a WER of 52.8% with CE-training (without MPE-training yet).

5. CONCLUSIONS

In this work we studied the effect of various LSTM hyperparameters. We demonstrated how to train deeper LSTM acoustic models with up to 10 layers. Important for this achievement was our introduction of pretraining for LSTMs which allows for such depth and is especially relevant for the deeper networks. We showed that we can reproduce good results with these findings on several different corpora and yield very good overall results which beats our best FFNN on Quaero by over 15% relatively. Given our current experience, we think that (N)Adam is always a good choice for optimization, some learning rate scheduling like Newbob is important, and pretraining helps, esp. for deeper models. Dropout together with L_2 regularization works best but should not be too high, and gradient noise often helps. First experiments with associative LSTMs were promising.

6. ACKNOWLEDGEMENTS

We thank Zoltán Tüske for the baseline FFNN Switchboard experiment and Pavel Golik for the Babel baseline experiment.

Partially supported by the Intelligence Advanced Research Projects Activity (IARPA) via Department of Defense U.S. Army Research Laboratory (DoD/ARL) contract no. W911NF-12-C-0012. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. Disclaimer: The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of IARPA, DoD/ARL, or the U.S. Government.

This research was partially supported by Ford Motor Company.

7. REFERENCES

[1] J. Schmidhuber, “Deep learning in neural networks: An overview,” *Neural Networks*, vol. 61, pp. 85–117, 2015, Published online 2014; based on TR arXiv:1404.7828 [cs.NE].

[2] Sepp Hochreiter and Jürgen Schmidhuber, “Long short-term memory,” *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[3] Haşim Sak, Andrew Senior, and Françoise Beaufays, “Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition,” *arXiv preprint arXiv:1402.1128*, 2014.

[4] Jürgen T Geiger, Zixing Zhang, Felix Weninger, Björn Schuller, and Gerhard Rigoll, “Robust speech recognition using long short-term memory recurrent neural networks for hybrid acoustic modelling,” in *INTERSPEECH*, 2014, pp. 631–635.

[5] “GitHub repository with config files for LSTM experiments in RETURNN,” `https://github.com/rwth-i6/returnn-experiments/tree/master/2016-lstm-paper`, 2016.

[6] Patrick Doetsch, Albert Zeyer, Paul Voigtlaender, Ilya Kulikov, Ralf Schlüter, and Hermann Ney, “RETURNN: The RWTH extensible training framework for universal recurrent neural networks,” *arXiv preprint arXiv:1608.00895, submitted to IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP) 2016*, 2016.

[7] Anthony J Robinson, “An application of recurrent nets to phone probability estimation,” *Neural Networks, IEEE Transactions on*, vol. 5, no. 2, pp. 298–305, 1994.

[8] Mike Schuster and Kuldip K Paliwal, “Bidirectional recurrent neural networks,” *IEEE Transactions on Signal Processing*, vol. 45, no. 11, pp. 2673–2681, 1997.

[9] Alex Graves, Navdeep Jaitly, and Abdel-rahman Mohamed, “Hybrid speech recognition with deep bidirectional LSTM,” in *Automatic Speech Recognition and Understanding (ASRU), 2013 IEEE Workshop on*. IEEE, 2013, pp. 273–278.

[10] Xiangang Li and Xihong Wu, “Constructing long short-term memory based deep recurrent neural networks for large vocabulary speech recognition,” in *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*. IEEE, 2015, pp. 4520–4524.

[11] Xiangang Li and Xihong Wu, “Improving long short-term memory networks using maxout units for large vocabulary speech recognition,” in *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*. IEEE, 2015, pp. 4600–4604.

[12] Tara N Sainath, Oriol Vinyals, Andrew Senior, and Hasim Sak, “Convolutional, long short-term memory, fully connected deep neural networks,” in *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*. IEEE, 2015, pp. 4580–4584.

[13] William Chan and Ian Lane, “Deep recurrent neural networks for acoustic modelling,” *arXiv preprint arXiv:1504.01482*, 2015.

[14] Andrew Senior, Hasim Sak, and Izhak Shafran, “Context dependent phone models for LSTM RNN acoustic modelling,” in *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*. IEEE, 2015, pp. 4585–4589.

[15] Yu Zhang, Guoguo Chen, Dong Yu, Kaisheng Yao, Sanjeev Khudanpur, and James Glass, “Highway long short-term memory RNNs for distant speech recognition,” *arXiv preprint arXiv:1510.08983*, 2015.

[16] Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio, “Empirical evaluation of gated recurrent neural networks on sequence modeling,” *CoRR*, vol. abs/1412.3555, 2014.

[17] Rafal Jozefowicz, Wojciech Zaremba, and Ilya Sutskever, “An empirical exploration of recurrent network architectures,” in *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, 2015, pp. 2342–2350.

[18] Klaus Greff, Rupesh Kumar Srivastava, Jan Koutník, Bas R Steunebrink, and Jürgen Schmidhuber, “LSTM: A search space odyssey,” *arXiv preprint arXiv:1503.04069*, 2015.

[19] Thomas M Breuel, “Benchmarking of LSTM networks,” *arXiv preprint arXiv:1508.02774*, 2015.

[20] Felix A Gers, Nicol N Schraudolph, and Jürgen Schmidhuber, “Learning precise timing with LSTM recurrent networks,” *The Journal of Machine Learning Research*, vol. 3, pp. 115–143, 2003.

[21] David Rybach, Stefan Hahn, Patrick Lehnen, David Nolden, Martin Sundermeyer, Zoltan Tüske, Simon Wiesler, Ralf Schlüter, and Hermann Ney, “RASR - the RWTH Aachen university open source speech recognition toolkit,” in *IEEE Automatic Speech Recognition and Understanding Workshop*, Waikoloa, HI, USA, Dec. 2011.

[22] Simon Wiesler, Alexander Richard, Pavel Golik, Ralf Schlüter, and Hermann Ney, “RASR/NN: The RWTH neural network toolkit for speech recognition,” in *IEEE International Conference on Acoustics, Speech, and Signal Processing*, Florence, Italy, May 2014, pp. 3313–3317.

[23] Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, James Bergstra, Ian J. Goodfellow, Arnaud Bergeron, Nicolas Bouchard, and Yoshua Bengio, “Theano: new features and speed improvements,” Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop, 2012.

[24] Markus Nußbaum-Thom, Simon Wiesler, Martin Sundermeyer, Christian Plahl, Stefan Hahn, Ralf Schlüter, and Hermann Ney, “The RWTH 2009 Quaero ASR evaluation system for English and German,” in *Interspeech*, Makuhari, Japan, Sept. 2010, pp. 1517–1520.

[25] Hervé Bourlard and Nelson Morgan, *Connectionist speech recognition: a hybrid approach*, vol. 247, Springer, 1994.

[26] Ralf Schlüter, L Bezrukov, Hannes Wagner, and Hermann Ney, “Gammatone features and feature combination for large vocabulary speech recognition,” in *Acoustics, Speech and Signal Processing, 2007. ICASSP 2007. IEEE International Conference on*. IEEE, 2007, vol. 4, pp. IV–649.

[27] Diederik Kingma and Jimmy Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014.

[28] Rupesh K Srivastava, Klaus Greff, and Jürgen Schmidhuber, “Training very deep networks,” in *Advances in Neural Information Processing Systems*, 2015, pp. 2368–2376.

[29] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, “Deep residual learning for image recognition,” *arXiv preprint arXiv:1512.03385*, 2015.

[30] Albert Zeyer, Ralf Schlüter, and Hermann Ney, “Towards online-recognition with deep bidirectional LSTM acoustic models,” in *Interspeech*, 2016.

[31] Alex Graves and Jürgen Schmidhuber, “Framewise phoneme classification with bidirectional lstm and other neural network architectures,” *Neural Networks*, vol. 18, no. 5, pp. 602–610, 2005.

[32] Boris T Polyak, “Some methods of speeding up the convergence of iteration methods,” *USSR Computational Mathematics and Mathematical Physics*, vol. 4, no. 5, pp. 1–17, 1964.

[33] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton, “On the importance of initialization and momentum in deep learning,” in *Proceedings of the 30th international conference on machine learning (ICML-13)*, 2013, pp. 1139–1147.

[34] Yurii Nesterov, “A method of solving a convex programming problem with convergence rate $O(1/k2)$,” in *Soviet Mathematics Doklady*, 1983, vol. 27, pp. 372–376.

[35] Simon Wiesler, Alexander Richard, Ralf Schlüter, and Hermann Ney, “Mean-normalized stochastic gradient for large-scale deep learning,” in *IEEE International Conference on Acoustics, Speech, and Signal Processing*, Florence, Italy, May 2014, pp. 180–184.

[36] Matthew D Zeiler, “Adadelta: An adaptive learning rate method,” *arXiv preprint arXiv:1212.5701*, 2012.

[37] John Duchi, Elad Hazan, and Yoram Singer, “Adaptive subgradient methods for online learning and stochastic optimization,” *The Journal of Machine Learning Research*, vol. 12, pp. 2121–2159, 2011.

[38] Timothy Dozat, “Incorporating Nesterov momentum into Adam,” Tech. Rep., Stanford University, 2015, `http://cs229.stanford.edu/proj2015/054_report.pdf`.

[39] A. Neelakantan, L. Vilnis, Q. V. Le, I. Sutskever, L. Kaiser, K. Kurach, and J. Martens, “Adding gradient noise improves learning for very deep networks,” *ArXiv preprint arXiv:1511.06807*, Nov. 2015.

[40] Tijmen Tieleman and Geoffrey Hinton, “Lecture 6.5 - RMSprop: Divide the gradient by a running average of its recent magnitude.,” COURSERA: Neural Networks for Machine Learning, 2012.

[41] Simon Funk, “RMSprop loses to SMORMS3 - beware the epsilon!,” `http://sifter.org/~simon/journal/20150420.html`, 2015.

[42] Caglar Gulcehre, Marcin Moczulski, and Yoshua Bengio, “Adasecant: robust adaptive secant method for stochastic gradient,” *arXiv preprint arXiv:1412.7419*, 2014.

[43] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov, “Improving neural networks by preventing co-adaptation of feature detectors,” *arXiv preprint arXiv:1207.0580*, 2012.

[44] Xavier Glorot and Yoshua Bengio, “Understanding the difficulty of training deep feedforward neural networks,” in *International Conference on Artificial Intelligence and Statistics*, 2010, pp. 249–256.

[45] Frank Seide, Gang Li, Xie Chen, and Dong Yu, “Feature engineering in context-dependent deep neural networks for conversational speech transcription,” in *Automatic Speech Recognition and Understanding (ASRU), 2011 IEEE Workshop on*. IEEE, 2011, pp. 24–29.

[46] Zoltán Tüske, Pavel Golik, Ralf Schlüter, and Hermann Ney, “Speaker adaptive joint training of gaussian mixture models and bottleneck features,” in *IEEE Automatic Speech Recognition and Understanding Workshop*, Scottsdale, AZ, USA, Dec. 2015, pp. 596–603.

[47] Ivo Danihelka, Greg Wayne, Benigno Uria, Nal Kalchbrenner, and Alex Graves, “Associative long short-term memory,” *arXiv preprint arXiv:1602.03032*, 2016.

[48] Pavel Golik, Zoltán Tüske, Ralf Schlüter, and Hermann Ney, “Multilingual features based keyword search for very low-resource languages,” in *Interspeech*, Dresden, Germany, Sept. 2015, pp. 1260–1264.