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

We present a comprehensive study of deep bidirectional long short-term memory (LSTM) recurrent neural network (RNN) based acoustic models for automatic speech recognition (ASR). We study the effect of size and depth and train models of up to 8 layers. We investigate the training aspect and study different variants of optimization methods, batching, truncated backpropagation, different regularization techniques such as dropout and L_2 regularization, and different gradient clipping variants.

The major part of the experimental analysis was performed on the Quaero corpus. Additional experiments also were performed on the Switchboard corpus. Our best LSTM model has a relative improvement in word error rate of over 14% compared to our best feed-forward neural network (FFNN) baseline on the Quaero task. On this task, we get our best result with an 8 layer bidirectional LSTM and we show that a pretraining scheme with layer-wise construction helps for deep LSTMs.

Finally we compare the training calculation time of many of the presented experiments in relation with recognition performance.

All the experiments were done with RETURNN, the RWTH extensible training framework for universal recurrent neural networks in combination with RASR, the RWTH ASR toolkit.

Index Terms: acoustic modeling, LSTM, RNN

1. Introduction

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 temporal sequences as in speech.

Only recently, it has been shown that LSTM based acoustic models (AM) outperform FFNNs on large vocabulary continuous speech recognition (LVCSR) [3, 4].

There are many aspects to be considered for training LSTMs which we are exploring in this work. Our experiments show that there is a huge variance in recognition performance depending on all the different aspects. Compared to our best FFNN baseline, we get a relative improvement in word error rate (WER) of over 14%. We also train deep LSTM networks with up to 8 layers for acoustic modeling and we discovered that a pretraining scheme can improve performance for deeper LSTMs. We are not aware of any previous work which applied pretraining for LSTMs in ASR.

2. Related work

Hybrid RNN-HMM models were developed in 1994 in [5]. One early hybrid LSTM-HMM was presented in [6] for TIMIT. [3, 4, 7, 8, 9, 10, 11, 12] 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.

Some LSTM related models like the Gated Recurrent Unit (GRU) were studied in [13, 14, 15, 16].

3. LSTM Model and Implementation

We use the standard LSTM model without peephole connections [17].

Our base tool is the RASR speech recognition toolkit [18, 19]. 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 [20, 21] 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 [20].

4. Comparisons

4.1. Corpus

We use a subset of 50 hours from the Quaero Broadcast Conversational English Speech database *train11* [22]. 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.

4.2. Baseline

We use the common NN-HMM hybrid acoustic model [23]. 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 [24]. We don't add any context window nor delta frames. 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 yields 15.3% WER on *eval10* and 20.3% WER on *eval11*.

Our minibatch construction is similar to e. g. [7] and described in detail in [20]. 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.

Note that our learning rate is not normalized in any way. Other software sometimes normalizes with the number of frames $\sum_i T_i$ or $T \cdot n_{\text{chunks}}$ so that the update step stays the same for every mini batch no matter how you change n_{chunks} or T. However, in our case, the total update scale per epoch stays the same no matter how one changes 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. 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 with an initial learning rate of 10^{-3}. We use gradient clipping of 10 by default.

4.3. 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 [25, 12] or deep residual learning [26]. 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 [12].

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 4.9, we get our overall best result as described in Section 4.10 with 8 layers, however we did not investigate the same setting for different number of 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 [26], 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. Dropout 0.1 + L_2, Adam, $n_{chunks} = 40$, WER on eval10. 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

4.4. Layer Size

In most experiments, we use a hidden layer size of 500 (i. e. 500 nodes 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 700, however a model with size 500 is much smaller and not so much worse, so we used that size for most other experiments.

Table 2: *Comparison of hidden layer size. 3 layers, dropout 0.1, Adadelta, $n_{chunks} = 20$. WER reported on eval10.*

layer size	#params[M]	WER[%]	epoch
300	7.9	15.8	30
500	18.7	15.1	15
700	34.0	**14.7**	15
1000	65.4	15.0	11

4.5. Topology: Bidirectional vs. Unidirectional

Our original experiment showed that we get quite a huge WER degradation (over 20% relative) with unidirectional LSTM networks compared to bidirectional ones.

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 [27].

4.6. 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 $n_{\text{chunks}} \in \{20, \dots, 80\}$ 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 higher because we see some of the data more often.

4.7. 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 [28, 29] where one variant only depends on the last minibatch (*mom*) and another variant depends on the full history (*mom2*), SGD with Nesterov momentum [30, 29], mean-normalized SGD (MNSGD) [31], Adadelta [32], Adagrad [33], Adam and Adamax [34], Adam without the learning rate decay term, Nadam (Adam with incorporated Nesterov momentum) [35], Adam with gradient noise [36], and Adam with MNSGD combined. We also tried Adasecant [37] but it did not converge in any of our experiments for this ASR task. We also test the effect of Newbob.

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 [20]. 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 different gradient clipping variants:

(*grad_clip*) We can clip the total gradient for all parameters.

(*upd_clip*) We can clip the update value for each parameter. Except for SGD, this is different than *grad_clip*.

(*err_clip*) We can clip the error signal in backpropagation after each time-frame.

(*grad_clip_z*) We can clip the error signal right after the loss function. This has the effect that the error signal of frames which are very miss-classified is limited. This can be interpreted as some kind of curriculum learning where the network itself characterizes the difficulty on frame level.

(*grad_discard_z*) We can expand on that idea and discard the error signal when it is higher than some threshold.

For method *grad_clip*, we found a value of 10 to be stable as well as good enough. No clipping yields the best performance in many cases except for a few and it sometimes does not even converge. A value of 2 is noticeably worse. These values are invariant of the learning rate. Method *upd_clip* is applied after we multiplied with the learning rate. A value of 0.1 gives a slight improvement but we did not further investigate this. Method *err_clip* was also not investigated in this paper although this is a common method.

The methods are again invariant to the learning rate. Method *grad_clip_z* gives us slight improvements with values 1 and 10 and value 0.1 did not converge. Method *grad_discard_z* just yielded the same WER for value 1 but we did not investigate this further.

Table 3: *Comparing different optimization methods. 3 layers, hidden layer size 500, dropout 0.1, $n_{chunks} = 20$. 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
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
MNSGD	10^{-4}	avg 0.5	20.2	18.2	17.8	20
		avg 0.995	19.1	16.8	16.4	18
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

4.8. Regularization Methods

We did many dropout experiments where we drop some nodes in forward connections. 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.

In initial experiments, dropout was always better than L_2 regularization. However, some later experiments showed that L_2 can also work as an alternative. Interestingly, the combination of both gives a big improvement and yields the best result. See Table 4.

4.9. Initialization and Pretraining

In all cases, we randomly initialize the parameters similar as suggested in [38].

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. We can then either train only the new layer (greedily) or the full network, where full network training usually was better.

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

In our initial experiments with not so optimal hyperparameters and not so deep networks, this pretraining scheme performed worse than not doing pretraining. In a later experiment with 5 layers, dropout + L_2 and Adam, we get a slight improvement from 13.6% WER to 13.5% WER.

For deeper networks, this 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 an 8 layer bidirectional LSTM.

Also, the training calculation time of the first few epochs is shorter.

4.10. Overall Best Model

So far we have analyzed many aspects in isolation. Often, when different methods and aspects are combined, we don't see the summed improvement as what we have seen for each individual method. We tried many combinations.

Our overall best model is an 8 layer bidirectional LSTM with 500 nodes, dropout 0.1 and L_2 10^{-2}, $n_{\text{chunks}} = 40$, $T = 50$, gradient noise 0.3, Nadam, no gradient clipping and the pretraining scheme as described in Section 4.9. This gives us a WER of 13.1% on *eval10* and 17.6% on *eval11*. We think that this pretraining was the most important aspect in this experiment, as the earlier results in Section 4.3 did not give good results for such deep networks.

4.11. Calculation Time vs. WER

We did over 250 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 5. 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.

4.12. 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 [39].

Table 5: *Total times until we get to a certain eval10 WER in a certain train epoch. If not specified we use just dropout.*

time	WER [%]	ep	model size	model details
2:18h	20.0	5	1x500	dropout + L_2
2:36h	17.2	5	3x300	time downsampling
3:40h	16.6	5	3x500	-
14:47h	13.9	13	3x500	dropout + L_2
23:15h	13.6	15	4x500	dropout + L_2
35:36h	13.2	18	5x500	dropout + L_2 + grad noise
52:24h	13.1	22	8x500	best as in Section 4.10

A good FFNN baseline yields a total WER of 19.1% (13.1% WER on SWB, 25.6% WER on CH). We trained a 5 layer bidir. LSTM with Nadam, gradient noise, dropout + L_2 and get a total WER of 17.1% (11.9% WER on SWB, 22.3% WER on CH). When we add one more layer and then retrain, we get a total WER of 16.7% (11.5% WER on SWB, 21.9% WER on CH). When we add an associative LSTM [40] layer instead, we get a total WER of 16.3% (11.1% on SWB, 21.6% on CH).

We also did a few experiments on Babel Javanese full language pack (`IARPA-babel402b-v1.0b`) which is a keyword-search (KWS) task (see [41] 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 bidir. LSTM with 19M parameters yields a WER of 52.8% with CE-training (without MPE-training yet).

5. Conclusions & Outlook

We outlined optimal LSTM hyperparameters such as the network depth and size, various training, optimization and regularization methods. We show that individual improvement findings can be combined and add up. We showed that we can reproduce good results with these findings on several different corpora and tasks and yield very good overall results which beats our best FFNN on Quaero by over 14% relatively.

We also demonstrated how to train deeper LSTM acoustic models with up to 8 layers which is more than what has been reported before in the literature. Important for this achievement was our introduction of pretraining for LSTMs.

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

[3] H. Sak, A. Senior, and F. Beaufays, "Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition," *arXiv preprint arXiv:1402.1128*, 2014, http://arxiv.org/pdf/1402.1128.

[4] J. T. Geiger, Z. Zhang, F. Weninger, B. Schuller, and G. Rigoll, "Robust speech recognition using long short-term memory recurrent neural networks for hybrid acoustic modelling," in *INTERSPEECH*, 2014, pp. 631–635.

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

[6] A. Graves, N. Jaitly, and A.-r. Mohamed, "Hybrid speech recognition with deep bidirectional LSTM," in *Automatic Speech Recognition and Understanding (ASRU), 2013 IEEE Workshop on*. IEEE, 2013, pp. 273–278, http://www.cs.toronto.edu/~graves/asru_2013.pdf.

[7] X. Li and X. 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.

[8] ——, "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.

[9] T. N. Sainath, O. Vinyals, A. Senior, and H. 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.

[10] W. Chan and I. Lane, "Deep recurrent neural networks for acoustic modelling," *arXiv preprint arXiv:1504.01482*, 2015.

[11] A. Senior, H. Sak, and I. 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.

[12] Y. Zhang, G. Chen, D. Yu, K. Yao, S. Khudanpur, and J. Glass, "Highway long short-term memory RNNs for distant speech recognition," *arXiv preprint arXiv:1510.08983*, 2015.

[13] J. Chung, Ç. Gülçehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *CoRR*, vol. abs/1412.3555, 2014. [Online]. Available: http://arxiv.org/abs/1412.3555

[14] R. Jozefowicz, W. Zaremba, and I. Sutskever, "An empirical exploration of recurrent network architectures," in *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, 2015, pp. 2342–2350.

[15] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber, "LSTM: A search space odyssey," *arXiv preprint arXiv:1503.04069*, 2015.

[16] T. M. Breuel, "Benchmarking of LSTM networks," *arXiv preprint arXiv:1508.02774*, 2015.

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

[18] D. Rybach, S. Hahn, P. Lehnen, D. Nolden, M. Sundermeyer, Z. Tüske, S. Wiesler, R. Schlüter, and H. Ney, "RASR - the RWTH Aachen university open source speech recognition toolkit," in *IEEE Automatic Speech Recognition and Understanding Workshop*, Waikoloa, HI, USA, Dec. 2011.

[19] S. Wiesler, A. Richard, P. Golik, R. Schlüter, and H. 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.

[20] P. Doetsch, A. Zeyer, P. Voigtlaender, I. Kulikov, R. Schlüter, and H. Ney, "RETURNN: The RWTH extensible training framework for universal recurrent neural networks," *work in preparation, can be requested from the authors*, 2016.

[21] F. Bastien, P. Lamblin, R. Pascanu, J. Bergstra, I. J. Goodfellow, A. Bergeron, N. Bouchard, and Y. Bengio, "Theano: new features and speed improvements," Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop, 2012.

[22] M. Nußbaum-Thom, S. Wiesler, M. Sundermeyer, C. Plahl, S. Hahn, R. Schlüter, and H. Ney, "The RWTH 2009 Quaero ASR evaluation system for English and German," in *Interspeech*, Makuhari, Japan, Sep. 2010, pp. 1517–1520.

[23] H. A. Bourlard and N. Morgan, *Connectionist speech recognition: a hybrid approach*. Springer, 1994, vol. 247, http://publications.idiap.ch/downloads/papers/2013/Bourlard_KLUWER_1994.pdf.

[24] R. Schlüter, L. Bezrukov, H. Wagner, and H. Ney, "Gammatone features and feature combination for large vocabulary speech recognition," in *Acoustics, Speech and Signal Processing, 2007. ICASSP 2007. IEEE International Conference on*, vol. 4. IEEE, 2007, pp. IV–649.

[25] R. K. Srivastava, K. Greff, and J. Schmidhuber, "Training very deep networks," in *Advances in Neural Information Processing Systems*, 2015, pp. 2368–2376.

[26] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *arXiv preprint arXiv:1512.03385*, 2015.

[27] A. Zeyer, R. Schlüter, and H. Ney, "Towards online-recognition with deep bidirectional LSTM acoustic models," in *Interspeech*, 2016.

[28] B. 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.

[29] I. Sutskever, J. Martens, G. Dahl, and G. 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.

[30] Y. Nesterov, "A method of solving a convex programming problem with convergence rate $O(1/k2)$," in *Soviet Mathematics Doklady*, vol. 27, no. 2, 1983, pp. 372–376.

[31] S. Wiesler, A. Richard, R. Schlüter, and H. 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.

[32] M. D. Zeiler, "Adadelta: An adaptive learning rate method," *arXiv preprint arXiv:1212.5701*, 2012, http://arxiv.org/abs/1212.5701.

[33] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *The Journal of Machine Learning Research*, vol. 12, pp. 2121–2159, 2011, http://dl.acm.org/citation.cfm?id=2021068.

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

[35] T. Dozat, "Incorporating Nesterov momentum into Adam," Stanford University, Tech. Rep., 2015, http://cs229.stanford.edu/proj2015/054_report.pdf.

[36] 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.

[37] C. Gulcehre, M. Moczulski, and Y. Bengio, "Adasecant: robust adaptive secant method for stochastic gradient," *arXiv preprint arXiv:1412.7419*, 2014.

[38] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *International Conference on Artificial Intelligence and Statistics*, 2010, pp. 249–256, http://jmlr.org/proceedings/papers/v9/glorot10a/glorot10a.pdf.

[39] Z. Tüske, P. Golik, R. Schlüter, and H. 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.

[40] I. Danihelka, G. Wayne, B. Uria, N. Kalchbrenner, and A. Graves, "Associative long short-term memory," *arXiv preprint arXiv:1602.03032*, 2016.

[41] P. Golik, Z. Tüske, R. Schlüter, and H. Ney, "Multilingual features based keyword search for very low-resource languages," in *Interspeech*, Dresden, Germany, Sep. 2015, pp. 1260–1264.