RASR/NN: THE RWTH NEURAL NETWORK TOOLKIT FOR SPEECH RECOGNITION

Simon Wiesler[1], *Alexander Richard*[1], *Pavel Golik*[1], *Ralf Schlüter*[1], *Hermann Ney*[1,2]

[1]Human Language Technology and Pattern Recognition,
Computer Science Department, RWTH Aachen University, Aachen, Germany
[2]LIMSI CNRS, Spoken Language Processing Group, Paris, France

ABSTRACT

This paper describes the new release of RASR - the open source version of the well-proven speech recognition toolkit developed and used at RWTH Aachen University. The focus is put on the implementation of the NN module for training neural network acoustic models. We describe code design, configuration, and features of the NN module. The key feature is a high flexibility regarding the network topology, choice of activation functions, training criteria, and optimization algorithm, as well as a built-in support for efficient GPU computing. The evaluation of run-time performance and recognition accuracy is performed exemplary with a deep neural network as acoustic model in a hybrid NN/HMM system. The results show that RASR achieves a state-of-the-art performance on a real-world large vocabulary task, while offering a complete pipeline for building and applying large scale speech recognition systems.

Index Terms— speech recognition, acoustic modeling, neural networks, GPU, open source, RASR

1. INTRODUCTION

Neural networks have become an essential part of automatic speech recognition (ASR) technology, in particular with the advent of deep learning since 2010, see for example [1]. Today, neural networks (NN) are not only in the focus of speech recognition research, but are also used by large companies such as Google, Microsoft and IBM [2].

At RWTH Aachen University, the speech recognition toolkit RASR has been developed and actively used for more than twelve years and been made open source in 2008 [3]. The idea behind making our speech recognition software publicly available is to simplify introduction to speech recognition research and to make scientific results reproducible. Other well-known open source speech recognition systems are CMU Sphinx [4], the HTK Toolkit [5], Julius [6], and more recently Kaldi [7].

The research leading to these results has received funding from the European Union Seventh Framework Programme EU-Bridge (FP7/2007-2013) under grant agreement N287658. The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 287755 (transLectures). H. Ney was partially supported by a senior chair award from DIGITEO, a French research cluster in Ile-de-France.

Previous versions of RASR had no support for neural networks. Instead, external tools were required. Most notably, the popular QuickNet software [8] from ICSI, Berkeley, could be used, or TNet [9] from Brno University of Technology. In this paper, we describe RASR's new neural network module NN for acoustic modeling. We compare RASR's performance with QuickNet being an efficient publicly available neural network software. While it is possible to build ASR systems by stacking various open source tools for neural network training and speech recognition, RASR covers the complete pipeline of speech processing, ranging from feature extraction over acoustic training to decoding, lattice processing and scoring. To illustrate the applicability of our software to a real-life task, we present experimental results on the English Quaero corpus, a large vocabulary continuous speech recognition (LVCSR) task.

The support of neural networks in an open source toolkit is challenging for two reasons. First, neural networks are a highly active research topic. It is not possible to support all recent advances in the stable version of RASR. We therefore wrote our neural network code base as generic as possible in order to allow for rapid integration of new concepts. This is achieved by decoupling different software parts as much as possible while maintaining clean interfaces that allow to modify and extend single aspects of neural networks. Second, the computation time required for training neural networks and using them in recognition is critical. Since our goal is to apply our methods to real-life tasks, the implementation must be very efficient. In our code, efficiency is achieved by supporting CPU multithreading and general purpose computing on GPUs. The GPU implementation makes use of the CUBLAS library and own CUDA kernels. For CPU computations, highly optimized matrix libraries following the BLAS standard can be used, for example Intel MKL or ACML.

RASR is available for download on our website[1]. The "RWTH ASR License" allows free usage including redistribution and modification for non-commercial use. The RASR website also offers comprehensive documentation as well as tutorials and recipes that help to rapidly develop own systems. Support is offered in form of a forum on the website.

[1]http://www-i6.informatik.rwth-aachen.de/rwth-asr

2. IMPLEMENTATION

In this section, we describe the main features of the neural network implementation in RASR.

2.1. Models

RASR supports feed-forward networks of a very general topology. The networks can have an arbitrary number of layers of different sizes and different activation functions. Each layer may have multiple inputs from other layers. In general, the network must be representable by a directed acyclic graph. In contrast, many other neural network implementations such as QuickNet only allow for a simple linear structure. The non-linear network structure enables for example the use of skip-connections or the joint training of hierarchical models [10]. Also, the extension of our software to recurrent networks is facilitated. The type of each layer can be chosen independently. Currently, RASR provides the commonly used activation functions $\mathrm{sigmoid}$, $\tanh$, linear and $\mathrm{softmax}$ as well as the recently proposed rectified linear units (rlu) [11].

Models can be stored either in binary or in XML format. While the binary format is more efficient, the XML format makes it easy to debug and analyze models as well as to import models from other neural network software.

2.2. Training

The structure of the neural network training code is similar to RASR's implementation of Gaussian mixture estimation. Various frame-level training criteria are implemented in separate trainer classes, which derive from a common base class `FeedForwardTrainer`. The trainer has a `NeuralNetwork`, an `Estimator`, a `Statistics` object, and a `Regularizer`. Given a mini-batch of training samples, the trainer computes the sufficient statistics based on a forward pass and an error backpropagation pass, updates the statistics, and performs an estimation step. If a GPU is available, all these operations are performed on GPU without synchronization to the main memory, avoiding expensive communication.

The available training criteria are *cross entropy*, *squared error*, and *binary divergence* [12]. Optionally, regularization parameters w.r.t. ℓ_2- or ℓ_1-norm, momentum and learning rates can be set individually for each layer.

While many neural network tools implement only stochastic gradient descent (SGD), RASR has a clean interface to the estimator. This simplifies the implementation of alternative optimization algorithms, which is currently an active research area for neural networks [13, 14]. Currently, the supported estimators include basic SGD, SGD with momentum, and a new stochastic second-order algorithm developed in our group [15]. In addition, batch estimation with gradient descent and Rprop [16] is possible. The estimator automatically creates a statistics object, which holds the sufficient statistics required for the optimization algorithm, e.g. the gradient for SGD.

Almost all training components can be configured independently. In particular, we do not only allow "natural pairings" of training criteria and output-layer, e.g. $\mathrm{softmax}$ with cross-entropy or $\mathrm{identity}$ with squared-error as in QuickNet. More exotic combinations like $\mathrm{softmax}$-outputs with squared-error are possible as well [17].

A difficulty in the implementation of stochastic optimization algorithms is that they require i.i.d. samples, which is simulated by shuffling the training data. For relatively small datasets, simply all training samples can be loaded into main memory and then accessed in random order. For larger datasets, we use a twofold randomization. We load as many training utterances as possible in random order into a buffer, and then shuffle the buffer on frame-level.

Another important aspect of stochastic optimization algorithms is the setting of learning rates. A simple and effective learning rate schedule is *Newbob*, as implemented in QuickNet. Newbob keeps the learning rate constant within each epoch. RASR also supports a *power schedule* that decays with every mini-batch and has been reported to perform slightly better than Newbob [18]. Mostly, we use a modification of Newbob which decays the learning rate less aggressively than the original Newbob [15].

2.3. Recognition

RASR comes with a dynamic decoder that is based on the history conditioned lexical tree approach [19]. The generic code structure of RASR makes it easy to implement a hybrid HMM/NN model in the decoder. The decoder uses a `FeatureScorer` for computing all required scores. The NN module offers a neural network feature scorer, which performs a forward pass of the network and converts the state posteriors to likelihood scores [20]. In case of a softmax output layer, the softmax activation function is not evaluated, because the normalization term is a constant in the search problem and can therefore be discarded. The feature scorer supports batching, i.e., multiple features are processed at once, which strongly speeds up matrix multiplications.

In tandem systems [21], the neural network is part of the feature extraction. Therefore, we added an efficient implementation of neural network forwarding to RASR's feature extraction module.

2.4. Feature extraction

RASR already provides a variety of signal analysis and preprocessing methods. Using RASR's *Flow* module, feature extraction methods can be defined as data flow graphs in XML format without writing much code in RASR. This makes it much easier to implement new signal analysis methods, which is an important part of research on neural networks for speech. Amongst others, RASR comes with Flow setups for MFCC, PLP, Gammatone, and MRASTA [22] features.

Table 1. Evaluation of QuickNet and RASR on a DNN with 493 inputs, six hidden layers with 2048 units each, and 4498 outputs on the Quaero task. Word error rates in %.

	Learning rate schedule			
	QuickNet		RASR	
Training using	Dev	Test	Dev	Test
QuickNet	19.6	26.2	19.4	25.9
RASR	19.8	25.7	19.2	25.4

As described in Subsection 2.2, stochastic optimization algorithms require that the features are loaded into a buffer and shuffled. Typical features for speech applications are obtained by stacking several consecutive frames in a sliding window. Buffering the windowed features directly thus increases the memory requirements proportional to the window size, which is typically in the order of 10 to 20. Instead, RASR can buffer only the central frames and construct the windowed features on-the-fly when generating the mini-batch.

2.5. Implementation details

RASR and its neural network module are entirely written in C++. Currently, the training can be performed using a single GPU. All parts of RASR/NN can be used in CPU mode as well. The RASR/NN module can be linked with highly optimized matrix libraries such as ACML or Intel MKL, which enable CPU multithreading.

With the use of batch optimization algorithms like Rprop, training can also be distributed across a large number of machines. It depends on the availability of resources and the amount of the training data whether stochastic or batch optimization algorithms are preferable.

The neural network code uses a template parameter for the floating point precision. According to our experience, stochastic algorithms do not require double precision, because the stochastic noise dominates rounding errors. Batch algorithms are much more sensitive to rounding error and might require double precision, e.g. for the statistics object.

The training of neural networks is designed such that a single run of the RASR binary performs one epoch of training. This has the advantage of greater flexibility of the training procedure. Not only one can monitor the performance by evaluating the intermediate model on some held-out set, store the intermediate results or calculate various statistics of the model, but also control network parameters: adjusting learning rate, varying number of layers, etc. This allows e.g. for an easy implementation of different flavors of pretraining [23].

3. EXPERIMENTS

In this section, we present experimental results that demonstrate the usefulness of RASR's neural network implementation on real-world tasks. We compare RASR/NN with QuickNet in order to verify the implementation.

Table 2. Effect of adding skip-connections to a neural network with one hidden layer of size 2048 units, and 4498 outputs. Frame error rates and word error rates in %.

	FER [%]	WER [%]	
Model topology	Train	Dev	Test
MLP	57.1	24.2	31.7
MLP + skip connections	54.7	23.5	30.8

3.1. Experimental setup

We performed experiments on the English Quaero corpus, a broadcast conversations recognition task. Our basic setup is the same as for our evaluation system for the Quaero project [24]. All models are trained on a 50-hour subset of the English Quaero training data, which allows for profiling training even without the use of a GPU. The development and test corpora (quaero-eval10 and quaero-eval11) consist of 3.7 and 3.3 hours of speech respectively.

For comparison, we also trained a conventional GMM-HMM system with VTLN-transformed MFCC features, according to the maximum likelihood criterion. 4498 context dependent states are modeled. In recognition, we use a lexicon with 150k words and a 4-gram language model. The GMM achieves 24.3% word error rate (WER) on the development data and 31.2% WER on the test data.

All neural networks have been trained according to the cross-entropy criterion with the context-dependent states as outputs. The training data corresponds to 17M frames, of which ten percent have been held out for cross-validation. The input to the neural networks is a 493-dimensional vector of VTLN-warped MFCC features and derivatives of first and second order which are computed in a sliding window of size 17. We evaluate two types of neural networks: a multi-layer perceptron (MLP) with one hidden layer of size 2048, and a *deep* neural network (DNN) with six hidden layers of the same size. DNN training has been initialized with a supervised layerwise pre-training as described in [23]. The SGD mini-batch size was set to 512. As mentioned in Subsection 2.2, RASR uses a modification of the QuickNet learning rate schedule *Newbob*. For a fair comparison we ran RASR and QuickNet with both learning rate configurations.

We aimed at keeping the QuickNet and RASR setups comparable at all levels. We used exactly the same training data and feature extraction for both implementations. The floating point precision of RASR has been set to single precision, which is also the hard-coded QuickNet setting. All recognitions were performed with RASR using the same search parameters.

3.2. Results

Table 1 shows the word error rates of RASR and QuickNet in experiments with DNNs. Although both implementations use the same algorithms, it is important to compare their perfor-

Table 3. Runtime analysis of RASR and QuickNet on neural networks with one and six hidden layers of size 2048, 493 inputs, and 4498 outputs. The training is performed on 50 hours of speech (approx. 15M frames). Runtime was measured on an Nvidia Tesla K20c (GPU) and a 12-core AMD processor (CPU).

Model	Hardware	Implementation	Time per mini-batch [ms]		Wallclock time [min]
			Forwarding	Backpropagation + SGD update	Training epoch
1x2048	GPU	QuickNet	37.0	49.3	36.0
		RASR	10.1	21.3	17.1
	CPU	QuickNet	193.1	1208.6	616.1
		RASR	120.2	236.0	187.0
6x2048	GPU	QuickNet	49.0	83.8	58.2
		RASR	23.8	46.3	37.1
	CPU	QuickNet	595.3	3499.5	1773.3
		RASR	330.4	715.4	549.3

mance, because the numerical stability of the code and implementation details impact the results.

The results in Table 1 confirm that the RASR learning rates are slightly better than the default Newbob learning rates in QuickNet. The Newbob learning rates performed worse on the development and test data, no matter whether we used RASR or QuickNet for training.

In three out of four experiments RASR achieved better results. The differences may not be significant, but we can conclude that the results obtained by RASR are competitive.

RASR allows to train models with a more general architecture than conventional N-layer networks. As a proof-of-concept, we trained an MLP and an MLP with the input layer being additionally connected to the output layer. As can be seen in Table 2, the skip-connections improve the recognition performance. Note however, that this model has more parameters and better results can also be achieved by increasing the number of layers.

3.3. Runtime analysis

In order to be able to train large models for real-world tasks and to tune systems, an efficient implementation is required. We performed a detailed runtime analysis of RASR and QuickNet. We measured the average computation time per mini-batch and determined the individual contributions of the forward pass and the backpropagation pass including the SGD parameter update. In addition, we measured the wall-clock time per epoch. The runtime analysis has been performed for the single-layer (MLP) and the six-layer networks (DNN) on an Nvidia Tesla K20c and a twelve-core AMD Opteron (2.3GHz). The results are summarized in Table 3.

Comparing the computation time per mini-batch in GPU mode, it can be seen that RASR is faster than QuickNet by a factor of 2.3 for the shallow network. The time measurement is consistent with the observed wall-clock time per epoch. The difference between both implementations is less pronounced for the deep network, but RASR is still 1.8 times faster than QuickNet.

A closer analysis revealed that QuickNet performs more data transfer than RASR. For example, the activations of the output layer are transferred to the main memory in order to compute frame error statistics on the CPU. Other factors may as well contribute to the slower training with QuickNet.

Comparing RASR's GPU and CPU implementations, the training on GPU is faster by a factor of 11 for the MLP and 15 for the DNN. The gain of using a GPU may be smaller on faster CPUs. Nevertheless, there is a clear advantage of using GPUs for training DNNs. The speed-up of RASR in comparison to QuickNet in CPU mode is even larger than in GPU mode.

4. CONCLUSION

We presented the new release of RASR with the NN module for training and evaluating neural networks for ASR. The presented open source software allows for a flexible configuration of network topology, choice of activation functions, training criteria and optimization algorithm. The computational efficiency is achieved by a transparent support of GPUs. The experimental evaluation on a real-world LVCSR task confirms that RASR achieves competitive recognition accuracy, while requiring far less computational time than QuickNet. With the NN module, RASR covers the complete ASR pipeline ranging from feature extraction over training to recognition, for both hybrid and tandem systems. Our future releases will include most recent improvements such as recurrent neural networks [25], convolutional layers [26], as well as sequence discriminative training [27].

5. ACKNOWLEDGMENT

We thank Christian Plahl for initial work on the neural network code, Zoltán Tüske for sharing his QuickNet expertise, and all former and current members of our group who have contributed or contribute to RASR.

6. REFERENCES

[1] A. Mohamed, G. Dahl, and G. Hinton, "Acoustic modeling using deep belief networks," *IEEE Trans. on Audio, Speech, and Language Processing*, no. 99, pp. 14–22, 2010.

[2] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *Signal Processing Magazine, IEEE*, vol. 29, no. 6, pp. 82–97, 2012.

[3] D. Rybach, C. Gollan, G. Heigold, B. Hoffmeister, J. Lööf, R. Schlüter, and H. Ney, "The RWTH Aachen university open source speech recognition system," in *Proc. Interspeech*, Brighton, UK, Sep. 2009, pp. 2111–2114.

[4] W. Walker, P. Lamere, P. Kwok, B. Raj, R. Singh, E. Gouvea, P. Wolf, and J. Woelfel, "Sphinx-4: a flexible open source framework for speech recognition," Sun Microsystems, Inc., Mountain View, CA, USA, Tech. Rep., 2004.

[5] S. Young, G. Evermann, M. Gales, T. Hain, D. Kershaw, X. Liu, G. Moore, J. Odell, D. Ollason, D. Povey, V. Valtchev, and P. Woodland, *The HTK book version 3.4*. Cambridge University Engineering Department, 2006.

[6] A. Lee, T. Kawahara, and K. Shikano, "Julius – an open source real-time large vocabulary recognition engine," in *Proc. Interspeech*, Aalborg, Denmark, Sep. 2001, pp. 1691–1694.

[7] D. Povey, A. Ghoshal, G. Boulianne, L. Burget, O. Glembek, N. Goel, M. Hannemann, P. Motlíček, Y. Qian, P. Schwarz, J. Silovský, G. Stemmer, and K. Veselý, "The Kaldi speech recognition toolkit," in *Proc. IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, Hawaii, USA, Dec. 2011.

[8] D. Johnson. (2004) QuickNet, speech group at ICSI, Berkeley. [Online]. Available: http://www.icsi.berkeley.edu/Speech/qn.html

[9] K. Veselý, L. Burget, and F. Grézl, "Parallel training of neural networks for speech recognition," in *Proc. Interspeech*, Makuhari, Japan, Sep. 2010, pp. 2934–2937.

[10] K. Veselý, M. Karafiát, and F. Grézl, "Convolutive bottleneck network features for LVCSR," in *Proc. IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, Hawaii, USA, Dec. 2011, pp. 42–47.

[11] V. Nair and G. E. Hinton, "Rectified linear units improve restricted Boltzmann machines," in *Proc. of the 27th Int. Conf. on Machine Learning*, Haifa, Israel, Jun. 2010, pp. 807–814.

[12] S. A. Solla, E. Levin, and M. Fleisher, "Accelerated learning in layered neural networks," *Complex Systems*, vol. 2, no. 6, pp. 625–639, Dec. 1988.

[13] J. Martens, "Deep learning via Hessian-free optimization," in *Proc. of the 27th Int. Conf. on Machine Learning*, vol. 951, 2010, p. 2010.

[14] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, Q. Le, M. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang, and A. Ng, "Large scale distributed deep networks," in *Advances in Neural Information Processing Systems 25*, 2012, pp. 1232–1240.

[15] S. Wiesler, A. Richard, R. Schlüter, and H. Ney, "Mean-normalized stochastic gradient for large-scale deep learning," in *(submitted to) Proc. IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, Florence, Italy, May 2014.

[16] M. Riedmiller and H. Braun, "A direct adaptive method for faster backpropagation learning: The RPROP algorithm," in *Proc. of the Int. Conf. on Neural Networks*, 1993, pp. 586–591.

[17] P. Golik, P. Doetsch, and H. Ney, "Cross-entropy vs. squared error training: a theoretical and experimental comparison," in *Proc. Interspeech*, Lyon, France, Aug. 2013, pp. 1756–1760.

[18] A. Senior, G. Heigold, M. Ranzato, and K. Yang, "An empirical study of learning rates in deep neural networks for speech recognition," in *Proc. IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, vol. 1, 2013, pp. 6724–6728.

[19] H. Ney and S. Ortmanns, "Progress in dynamic programming search for LVCSR," *Proc. of the IEEE*, vol. 88, no. 8, pp. 1224–1240, Aug. 2000.

[20] H. A. Bourlard and N. Morgan, *Connectionist speech recognition: a hybrid approach*. Norwell, MA, USA: Kluwer Academic Publishers, 1993.

[21] H. Hermansky, D. Ellis, and S. Sharma, "Tandem connectionist feature extraction for conventional HMM systems," in *Proc. IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, vol. 3, Istanbul, Turkey, Jun. 2000, pp. 1635–1638.

[22] H. Hermansky and P. Fousek, "Multi-resolution RASTA filtering for TANDEM-based ASR," in *Proc. Interspeech*, Lisbon, Portugal, Sep. 2005, pp. 361–364.

[23] F. Seide, G. Li, X. Chen, and D. Yu, "Feature engineering in context-dependent deep neural networks for conversational speech transcription," in *Proc. IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, Hawaii, USA, Dec. 2011, pp. 24–29.

[24] M. Sundermeyer, M. Nußbaum-Thom, S. Wiesler, C. Plahl, A. E. Mousa, S. Hahn, D. Nolden, R. Schlüter, and H. Ney, "The RWTH 2010 QUAERO ASR evaluation system for English, French, and German," in *Proc. IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, Prague, Czech, May 2011, pp. 2212–2215.

[25] A. Graves, N. Jaitly, and A.-r. Mohamed, "Hybrid speech recognition with deep bidirectional LSTM," in *Proc. IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, Olomouc, Czech Republic, Dec. 2013, pp. 273–278.

[26] O. Abdel-Hamid, A. Mohamed, H. Jiang, and G. Penn, "Applying convolutional neural networks concepts to hybrid NN-HMM model for speech recognition," in *Proc. IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, Kyoto, Japan, Mar. 2012, pp. 4277–4280.

[27] B. Kingsbury, "Lattice-based optimization of sequence classification criteria for neural-network acoustic modeling," in *Proc. IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, Taipei, Taiwan, Apr. 2009, pp. 3761–3764.