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

Simon Wiesler¹, Alexander Richard¹, Pavel Golik¹, Ralf Schlüter¹, Hermann Ney¹,²

¹Human Language Technology and Pattern Recognition, Computer Science Department, RWTH Aachen University, Aachen, Germany
²LIMSI 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 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].

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 website¹. 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.

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 (tra$Lectures). H. Ney was partially supported by a senior chair award from DIGITEO, a French research cluster in Ile-de-France.

¹http://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 sigmoid, tanh, linear and softmax as well as the recently proposed rectified linear units (relu) [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 FeedForward Trainer. 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. \( \ell_2 \)- or \( \ell_1 \)-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. softmax with cross-entropy or identity with squared-error as in QuickNet. More exotic combinations like 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 %.

<table>
<thead>
<tr>
<th>Model topology</th>
<th>FER [%]</th>
<th>WER [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLP</td>
<td>57.1</td>
<td>24.2</td>
</tr>
<tr>
<td>MLP + skip connections</td>
<td>54.7</td>
<td>23.5</td>
</tr>
</tbody>
</table>

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 can one 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 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).

<table>
<thead>
<tr>
<th>Model</th>
<th>Hardware</th>
<th>Implementation</th>
<th>Time per mini-batch [ms]</th>
<th>Wallclock time [min]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Forwarding</td>
<td>Backpropagation + SGD update</td>
</tr>
<tr>
<td>1x2048</td>
<td>GPU</td>
<td>QuickNet</td>
<td>37.0</td>
<td>49.3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RASR</td>
<td>10.1</td>
<td>21.3</td>
</tr>
<tr>
<td></td>
<td>CPU</td>
<td>QuickNet</td>
<td>193.1</td>
<td>1208.6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RASR</td>
<td>120.2</td>
<td>236.0</td>
</tr>
<tr>
<td>6x2048</td>
<td>GPU</td>
<td>QuickNet</td>
<td>49.0</td>
<td>83.8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RASR</td>
<td>23.8</td>
<td>46.3</td>
</tr>
<tr>
<td></td>
<td>CPU</td>
<td>QuickNet</td>
<td>595.3</td>
<td>3499.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RASR</td>
<td>330.4</td>
<td>715.4</td>
</tr>
</tbody>
</table>

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


