

WAV2LETTER++: THE FASTEST OPEN-SOURCE SPEECH RECOGNITION SYSTEM

Vineel Pratap, Awni Hannun, Qiantong Xu, Jeff Cai, Jacob Kahn, Gabriel Synnaeve,
Vitaliy Liptchinsky, Ronan Collobert

Facebook AI Research

ABSTRACT

This paper introduces wav2letter++, the fastest open-source deep learning speech recognition framework. wav2letter++ is written entirely in C++, and uses the ArrayFire tensor library for maximum efficiency. Here we explain the architecture and design of the wav2letter++ system and compare it to other major open-source speech recognition systems. In some cases wav2letter++ is more than $2\times$ faster than other optimized frameworks for training end-to-end neural networks for speech recognition. We also show that wav2letter++’s training times scale linearly to 64 GPUs, the highest we tested, for models with 100 million parameters. High-performance frameworks enable fast iteration, which is often a crucial factor in successful research and model tuning on new datasets and tasks.

Index Terms— speech recognition, open source software, end-to-end

1. INTRODUCTION

With the growing interest in automatic speech recognition (ASR), the open-source software ecosystem has seen a proliferation of ASR systems and toolkits, including Kaldi [1], ESPNet [2], OpenSeq2Seq [3] and Eesen[4]. Over the last decade these frameworks have shifted from traditional speech recognition based on Hidden Markov Models (HMM) and Gaussian Mixture Models (GMM) to end-to-end neural network based systems. Many of the recent open-source ASR toolkits, including the one presented in this paper, rely on end-to-end acoustic modeling based on graphemes rather than phonemes. The reason for this shift is two-fold: end-to-end models are significantly simpler and the gap in accuracy to HMM/GMM systems is rapidly closing. C++ is the 3rd most popular programming language in the world¹. It allows for complete resource control for high-performance and mission critical systems, and, in addition, static typing helps with large-scale projects by catching any contract mismatches at compile time. Moreover, native libraries can be easily invoked from virtually any programming language. However, adoption of C++ in the machine learning community

has been stalled by the absence of well-defined C++ APIs in mainstream frameworks, and C++ was used mostly for performance critical components. As a code base becomes larger, it also becomes cumbersome and error-prone to switch back and forth between a scripting language and C++. Also, provided the adequate libraries, developing in modern C++ is not much slower than in a scripting language. In this paper, we introduce the first open-source speech recognition system written completely in C++. By using modern C++, we do not sacrifice ease of programming yet maintain the ability to write highly efficient and scalable software. In this work, we focus on the technical aspects of ASR systems, such as training and decoding speed, and scalability. The rest of this paper is structured as follows. In Section 2, we discuss the design of wav2letter++. In Section 3, we briefly discuss other existing major open-source systems, and benchmark their performance against ours in Section 4.

2. DESIGN

The design of wav2letter++ is motivated by three requirements. First, the toolkit must be able to efficiently train models on datasets containing many thousands of hours of speech. Second, expressing and incorporating new network architectures, loss functions, and other core operations should be simple. And third, the path from model research to deployment should be straightforward, requiring as little new code as possible while maintaining the flexibility needed for research.

2.1. ArrayFire Tensor Library

We use ArrayFire [5] as our primary library for tensor operations. We chose ArrayFire for several reasons. ArrayFire is a highly optimized tensor library that can execute on multiple back-ends including a CUDA GPU back-end and a CPU back-end. ArrayFire also uses just-in-time code generation to combine series of simple operations into a single kernel call. This results in faster execution for memory bandwidth bound operations and can reduce peak memory use. Another important feature of ArrayFire is the simple interface for constructing and operating on arrays. Compared to other C++ tensor libraries which also support CUDA, the ArrayFire interface is less verbose and relies on fewer C++ idiosyncrasies.

¹<https://www.tiobe.com/tiobe-index/>

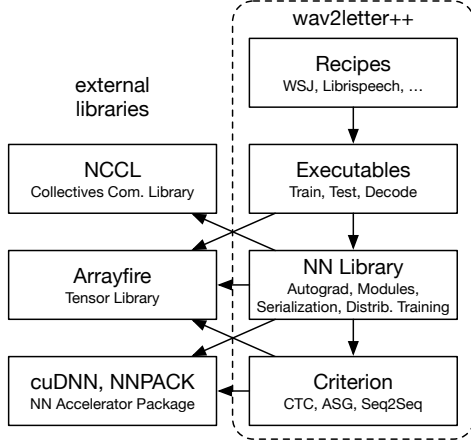


Fig. 1. The wav2letter++ library architecture.

2.2. Data Preparation and Feature Extraction

Our feature extraction supports multiple audio file formats (e.g. wav, flac... / mono, stereo / int, float) and several feature types including the raw audio, a linearly scaled power spectrum, log-Mels (MFSC) and MFCCs. We use the FFTW library to compute discrete Fourier transforms [6]. Data loading in wav2letter++ computes features on the fly prior to each network evaluation. This makes exploring alternative features simpler, allows for dynamic data augmentation and makes deploying models much easier since the full end-to-end pipeline can run from a single binary. To make this efficient while training models, we load and decode the audio and compute the features asynchronously and in parallel. For the models and batch sizes we tested, the time spent in data loading is negligible.

2.3. Models

We support several end-to-end sequence models. Every model is divided into a network and criterion. The network is a function of just the input whereas the criterion is a function of both the input and the target transcription. While the network always has parameters, the parameters of the criterion are optional. This abstraction allows us to easily train different models with the same training pipeline. The supported criteria include Connectionist Temporal Classification (CTC) [7], the original wav2letter AutoSegCriterion (ASG) [8], and Sequence-to-Sequence models with attention (S2S) [9, 10]. The CTC criterion does not have parameters whereas the ASG and S2S criteria both have parameters which can be learned. Furthermore, we note that adding new sequence criteria is particularly easy given that loss functions like ASG and CTC can be efficiently implemented in C++. We support a wide range of network architectures and activation functions – too many to list here. For certain operations

```
Variable forward(const Variable& x) {
    auto hidden = matmul(weights[0], x);
    hidden = max(hidden, 0); // ReLU
    return matmul(weights[1], hidden);
}
Variable criterion(const Variable& yhat,
                  const Variable& y) {
    auto probs = sigmoid(yhat);
    return -(y * log(probs) +
            (1 - y) * log(1 - probs));
}
for (const auto& xy : trainSet) {
    criterion(forward(xy[0]), xy[1]).backward();
    for (auto& w : weights) {
        w -= lr * w.grad();
        w.zeroGrad();
    }
}
```

Fig. 2. Example: one hidden layer MLP trained with binary cross-entropy and SGD, using automatic differentiation.

we extend the core ArrayFire CUDA back-end with more efficient cuDNN operations [11]. We use the 1D and 2D convolutions and the RNN routines provided by cuDNN, among others. Since the network library we use provides dynamic graph construction and automatic differentiation, building new layers or other primitive operations requires little effort. We give an example showing how to build and train a one layer MLP with the binary cross-entropy loss (in Fig. 2) to demonstrate the simplicity of the C++ interface.

2.4. Training and Scale

Our training pipeline gives maximum flexibility for the user to experiment with different features, architectures and optimization parameters. Training can be run in three modes - `train` (flat-start training), `continue` (continuing with a checkpoint state), and `fork` (for e.g. transfer learning). We support standard optimization algorithms including SGD and other commonly used first-order gradient-based optimizers. We scale wav2letter++ to larger datasets with data-parallel, synchronous SGD. For inter-process communication we use the NVIDIA Collective Communication Library (NCCL2)². To minimize wait time between processes and improve the efficiency of a single process, we sort the dataset on input length prior to constructing batches for training [12].

2.5. Decoding

The wav2letter++ decoder is a beam-search decoder with several optimizations to improve efficiency [13]. We use the same decoding objective as [13], which includes the constraint from a language model and a word insertion term.

²<https://github.com/NVIDIA/nccl>

Name	Language	Model(s)	ML Syst.
Kaldi	C++, Bash	HMM/GMM DNN/LF-MMI	- -
ESPNet	Python, Bash	CTC, seq2seq, hybrid	PyTorch, Chainer
OpenSeq2Seq	Python, C++	CTC, seq2seq	TensorFlow
wav2letter++	C++	CTC, seq2seq, ASG	ArrayFire

Table 1. Major open-source speech recognition systems.

The decoder interface accepts as input the emissions and (if relevant) transitions from the acoustic model. We also give the decoder a Trie which contains the word dictionary and a language model. We support any type of language model which exposes the interface required by our decoder including n-gram LMs and any other stateless parametric LM. We provide a thin wrapper on top of KenLM for n-gram language models [14].

3. RELATED WORK

We give a brief overview of other commonly used open-source speech recognition systems, including Kaldi [1], ESPNet [2], and OpenSeq2Seq [3]. The Kaldi Speech Recognition Toolkit [1] is by far the oldest of the aforementioned and consists of a set of stand-alone command-line tools. Kaldi supports HMM/GMM and hybrid HMM/NN-based acoustic modeling, and includes phone-based recipes. End-to-End Speech Processing Toolkit (ESPNet) [2] tightly integrates with Kaldi and uses it for feature extraction and data pre-processing. ESPNet uses Chainer [15] or PyTorch [16] as a back-end to train acoustic models. It is mostly written in Python, however, following the style of Kaldi, high-level work-flows are expressed in bash scripts. While encouraging the decoupling of system components, this approach lacks the benefit of statically-typed object-oriented programming languages in expressing type-safe, readable and intuitive interfaces. ESPNet features both CTC-based [7] and attention-based encoder-decoder [10] implementations as well as a hybrid model combining both criteria. OpenSeq2Seq [3], similarly to ESPNet, features both CTC-based and encoder-decoder models and is written in Python, using TensorFlow [17] rather than PyTorch as the back-end. For high-level workflows OpenSeq2Seq also relies on bash scripts that call Perl and Python scripts. A notable feature of the OpenSeq2Seq system is its support for mixed-precision training. Also, both ESPNet and OpenSeq2Seq support models for Text-To-Speech (TTS). Table 1 depicts the taxonomy of these open-source speech processing systems. As the table shows, wav2letter++ is the only framework written entirely in C++, which (i) enables easy integration into existing applications implemented virtually in any programming language;

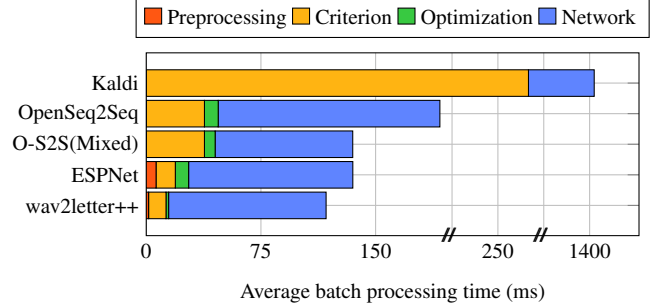


Fig. 3. Time in milliseconds for the major steps in the training loop. The times are averaged for each batch over a full epoch.

(ii) better supports large-scale development with static typing and object oriented programming; (iii) allows for maximum efficiency as discussed in Section 4. In contrast, dynamically-typed languages such as Python promote quick prototyping, but the lack of enforced static typing often hinders large-scale development.

4. EXPERIMENTS

In this section we discuss the performance of ESPNet, Kaldi, OpenSeq2Seq and wav2letter++ in a comparative study. The ASR systems are evaluated on the large vocabulary task of the Wall Street Journal (WSJ) dataset [18]. We measure both the average epoch time on WSJ during training and the average utterance decoding latency. The machines we use for the experiments have the following hardware configuration: each machine features eight NVIDIA Tesla V100 Tensor Core GPUs on NVIDIA SXM2 Modules with 16GB of memory. Each compute node has 2 Intel Xeon E5-2698 v4 CPUs, totalling 40 (2×20) cores, 80 hardware threads (“cores”), at 2.20GHz. All machines are connected over a 100Gbps InfiniBand network.

4.1. Training

We evaluate training time with respect to both scaling network parameters and increasing the number of GPUs used. We consider 2 types of neural network architectures: recurrent, with 30 million parameters, and purely convolutional, with 100 million parameters, as depicted in the top and bottom charts of Figure 4, respectively. For OpenSeq2Seq we consider both float32 as well as mixed precision float16 training. For both networks, we use 40-dimensional log-mel filterbanks as inputs, and CTC [7] as the criterion (CPU-based implementation). For Kaldi, we use the LF-MMI [19] criterion as CTC training is not available in the standard Kaldi recipes. All models are trained with SGD with momentum. We use a batch size of 4 utterances per GPU. Every run is restricted to using 5 CPU cores for each GPU. Figure 3 provides more detailed look into major components of the

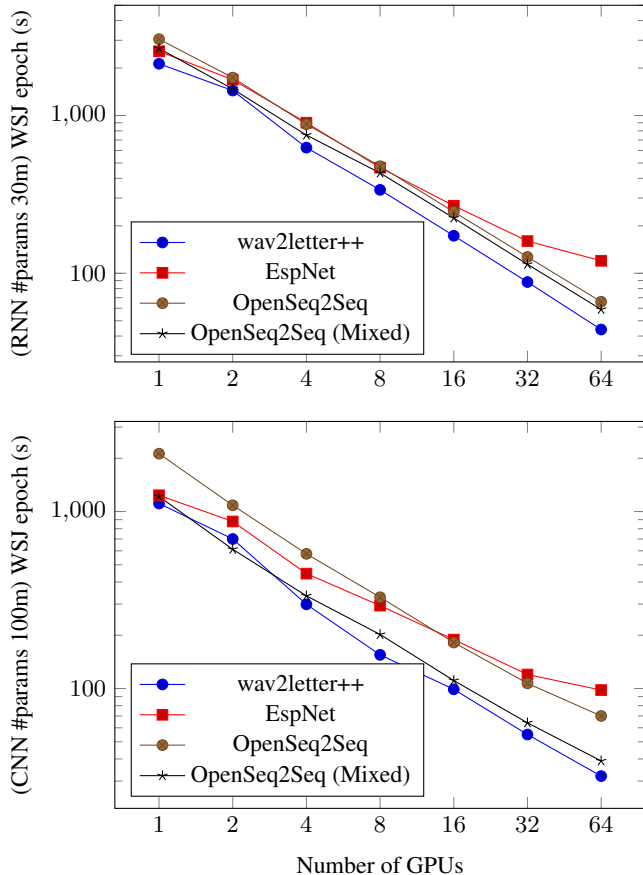


Fig. 4. Comparison of training times (log scale). **Top:** An RNN with 30m parameters, inspired by DeepSpeech 2 [12]: 2 spatial convolution layers, followed by 5 bidirectional LSTM layers, followed by 2 linear layers. **Bottom:** A CNN with 100m parameters, similar to [13]: 18 temporal convolution layers, followed by 1 linear layer.

training pipeline; the processing time is averaged over entire epoch using a single GPU. For both models, wav2letter++ has a clear advantage that increases as we scale out the computation. For smaller models with 30 million parameters, wav2letter++ is more than 15% faster than the next-best system, even on a single GPU. Note that since we use 8 GPU machines, experiments on 16, 32 and 64 GPUs involve multi-node communication. ESPNet did not support multi-node training out-of-the-box. We extend it by using the PyTorch `DistributedDataParallel` module with the NCCL2 back-end. ESPNet relies on pre-computed input features, while wav2letter++ and OpenSeq2Seq compute features on the fly for the sake of flexibility. In some cases, mixed precision training decreases the epoch time by more than 1.5x for OpenSeq2Seq. This is an optimization which wav2letter++ can benefit from in the future. The Kaldi recipe for LF-MMI does not synchronize gradients for each SGD update; the per-epoch time is still more than 20x slower. We did not include

Name	WER (%)	Time/sample (ms)	Memory (GB)
ESPNet	7.20	1548	–
OpenSeq2Seq	5.00	1700	7.8
OpenSeq2Seq	4.92	9500	26.6
wav2letter++	5.00	10	3.9
wav2letter++	4.91	140	5.5

Table 2. Decoding performance on LibriSpeech *dev-clean*.

Kaldi in Figure 4 as the criterion (LF-MMI) and optimization algorithm are not easily comparable.

4.2. Decoding

wav2letter++ includes a one-pass beam-search decoder (see Section 2.5), written in C++. We benchmark it against other beam-search decoders available in OpenSeq2Seq and ESPNet. Kaldi is not included, as it does not support CTC decoding, and implements a WFST-based decoder. We feed each decoder identical, pre-computed emissions generated by a fully-convolutional OpenSeq2Seq model Wave2Letter+³ trained on LibriSpeech. This enables independent measurement of performance given the same model. The 4-gram LibriSpeech language model is used for OpenSeq2Seq and wav2letter++, as ESPNet does not support n-gram LM decoding. In Table 2, we report decoding time and peak memory usage, for single thread decoding, on LibriSpeech *dev-clean* to reach a WER of 5.0%, and the best-obtainable WER for each framework. Hyper-parameters were heavily tuned such that reported results reflect the best-possible speed for the reported WER. wav2letter++ not only outperforms similar decoders by more than an order of magnitude, but also uses considerably less memory.

5. CONCLUSION

In this paper we introduced wav2letter++: a fast and simple system for developing end-to-end speech recognizers. The framework is written entirely in C++ which makes it efficient to train models and perform real-time decoding. Our initial implementation shows promising results compared to the other speech frameworks; though wav2letter++ can continue to benefit from further optimization. Because of its simple and extensible interface, wav2letter++ is well suited as a platform for rapid research in end-to-end speech recognition. At the same time, we leave open the possibility that certain optimizations might be possible with Python-based ASR systems to narrow the gap with wav2letter++.

³<https://nvidia.github.io/OpenSeq2Seq/html/speech-recognition.html>

6. REFERENCES

- [1] Daniel Povey, Arnab Ghoshal, Gilles Boulianne, Lukas Burget, Ondrej Glembek, Nagendra Goel, Mirko Hannemann, Petr Motlicek, Yanmin Qian, Petr Schwarz, et al., “The kaldi speech recognition toolkit,” in *IEEE 2011 workshop on automatic speech recognition and understanding*. IEEE Signal Processing Society, 2011, number EPFL-CONF-192584.
- [2] Shinji Watanabe, Takaaki Hori, Shigeki Karita, Tomoki Hayashi, Jiro Nishitoba, Yuya Unno, Nelson Enrique Yalta Soplin, Jahn Heymann, Matthew Wiesner, Nanxin Chen, et al., “Espnet: End-to-end speech processing toolkit,” *arXiv preprint arXiv:1804.00015*, 2018.
- [3] Oleksii Kuchaiev, Boris Ginsburg, Igor Gitman, Vitaly Lavrukhin, Carl Case, and Paulius Micikevicius, “Openseq2seq: extensible toolkit for distributed and mixed precision training of sequence-to-sequence models,” *arXiv preprint arXiv:1805.10387*, 2018.
- [4] Yajie Miao, Mohammad Gowayyed, and Florian Metze, “Eesen: End-to-end speech recognition using deep rnn models and wfst-based decoding,” in *Automatic Speech Recognition and Understanding (ASRU), 2015 IEEE Workshop on*. IEEE, 2015, pp. 167–174.
- [5] James Malcolm, Pavan Yalamanchili, Chris McClanahan, Vishwanath Venugopalakrishnan, Krupal Patel, and John Melonakos, “Arrayfire: a gpu acceleration platform,” 2012.
- [6] Matteo Frigo and Steven G. Johnson, “The design and implementation of FFTW3,” *Proceedings of the IEEE*, vol. 93, no. 2, pp. 216–231, 2005, Special issue on “Program Generation, Optimization, and Platform Adaptation”.
- [7] Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber, “Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks,” in *Proceedings of the 23rd international conference on Machine learning*. ACM, 2006, pp. 369–376.
- [8] Ronan Collobert, Christian Puhersch, and Gabriel Synnaeve, “Wav2letter: an end-to-end convnet-based speech recognition system,” *CoRR*, vol. abs/1609.03193, 2016.
- [9] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio, “Neural machine translation by jointly learning to align and translate,” *arXiv preprint arXiv:1409.0473*, 2014.
- [10] Jan K Chorowski, Dzmitry Bahdanau, Dmitriy Serdyuk, Kyunghyun Cho, and Yoshua Bengio, “Attention-based models for speech recognition,” in *Advances in neural information processing systems*, 2015, pp. 577–585.
- [11] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer, “cudnn: Efficient primitives for deep learning,” *arXiv preprint arXiv:1410.0759*, 2014.
- [12] Awni Hannun, Carl Case, Jared Casper, Bryan Catanzaro, Greg Diamos, Erich Elsen, Ryan Prenger, Sanjeev Satheesh, Shubho Sengupta, Adam Coates, et al., “Deep speech: Scaling up end-to-end speech recognition,” *arXiv preprint arXiv:1412.5567*, 2014.
- [13] Vitaliy Liptchinsky, Gabriel Synnaeve, and Ronan Collobert, “Letter-based speech recognition with gated convnets,” *CoRR*, vol. abs/1712.09444, 2017.
- [14] Kenneth Heafield, “Kenlm: Faster and smaller language model queries,” in *Proceedings of the Sixth Workshop on Statistical Machine Translation*. Association for Computational Linguistics, 2011, pp. 187–197.
- [15] Seiya Tokui, Kenta Oono, Shohei Hido, and Justin Clayton, “Chainer: a next-generation open source framework for deep learning,” in *Proceedings of workshop on machine learning systems (LearningSys) in the twenty-ninth annual conference on neural information processing systems (NIPS)*, 2015, vol. 5, pp. 1–6.
- [16] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer, “Automatic differentiation in pytorch,” 2017.
- [17] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al., “Tensorflow: a system for large-scale machine learning,” in *OSDI*, 2016, vol. 16, pp. 265–283.
- [18] Douglas B Paul and Janet M Baker, “The design for the wall street journal-based csr corpus,” in *Proceedings of the workshop on Speech and Natural Language*. Association for Computational Linguistics, 1992, pp. 357–362.
- [19] Daniel Povey, Vijayaditya Peddinti, Daniel Galvez, Pegah Ghahremani, Vimal Manohar, Xingyu Na, Yiming Wang, and Sanjeev Khudanpur, “Purely sequence-trained neural networks for asr based on lattice-free mmi,” in *Interspeech*, 2016, pp. 2751–2755.