OpenSeq2Seq: Extensible Toolkit for Distributed and Mixed Precision Training of Sequence-to-Sequence Models

Oleksii Kuchaiev, Boris Ginsburg, Igor Gitman,
Vitaly Lavrukhin, Carl Case, Paulius Micikevicius
NVIDIA
Santa Clara, CA
{okuchaiev,bginsburg,igitman}@nvidia.com
{vlavrukhin,carlc,pauliusm}@nvidia.com

Abstract

We present OpenSeq2Seq – an open-source toolkit for training sequence-to-sequence models. The main goal of our toolkit is to allow researchers to most effectively explore different sequence-to-sequence architectures. The efficiency is achieved by fully supporting distributed and mixed-precision training.

OpenSeq2Seq provides building blocks for training encoder-decoder models for neural machine translation and automatic speech recognition. We plan to extend it with other modalities in the future.

1 Introduction

Sequence-to-Sequence models built around the encoder-decoder paradigm (Sutskever et al., 2014) have been successfully used for natural language processing (NLP) (Vaswani et al., 2017), image-captioning (Xu et al., 2015), and automatic speech recognition (ASR) (Chan et al., 2015; Battenberg et al., 2017). However, implementing a sequence-to-sequence model in a general purpose deep learning framework such as TensorFlow (Abadi et al., 2016), CNTK (Yu et al., 2014) or PyTorch (Paszke et al., 2017) can be challenging, especially with support for distributed training. Several open-source toolkits have been proposed in recent years in an attempt to tackle this challenge. Among the most popular ones are: OpenNMT (Klein et al., 2017), Seq2Seq (Britz et al., 2017), NMT (Luong et al., 2017), and Tensor2Tensor (Vaswani et al., 2017). These toolkits make it much easier to reproduce most current state-of-the-art results and train your own models on new datasets. OpenSeq2Seq is inspired by these approaches with an additional focus on distributed and mixed-precision training.

In particular, OpenSeq2Seq adds support for mixed precision training as described in (Micikevicius et al., 2017). It uses the IEEE float16 data format to reduce memory requirements and speed up training on modern deep learning hardware such as NVIDIA's Volta GPUs. Furthermore, OpenSeq2Seq supports multi-GPU and multi-node distributed training.

OpenSeq2Seq is built using TensorFlow and is available at: https://github.com/NVIDIA/OpenSeq2Seq.

2 Design

The main design goals of OpenSeq2Seq are extensibility and modularity. It provides several core abstract classes which users can inherit from when adding new models: `DataLayer`, `Model`, `Encoder`, `Decoder` and `Loss`. The `Model` class implements distributed and mixed precision training support. For distributed training we support two modes, both following data-parallel approaches with synchronous updates: (1) multi-tower mode in which a separate TensorFlow graph is built on every GPU and (2) Horovod-based mode (Sergeev and Del Balso, 2018) which allows both *multi-node* as well as *multi-GPU* executions.

At a high level, the `Encoder` is a model block which consumes data and produces a representation; while the `Decoder` is a model block which consumes a representation and produces data and/or output. While we do not strictly enforce this, we assume that any encoder can be combined with any decoder, thus improving flexibility and simplicity of experimentation. Note that it is possible to have a model consisting of only an encoder, only a decoder, or having more than one encoder and/or decoder. Currently, we provide the DataLayer, Encoder, Decoder and Loss class implementations for neural machine trans-

Proceedings of Workshop for NLP Open Source Software, pages 41–46
Melbourne, Australia, July 20, 2018.

lation (NMT) and automatic speech recognition (ASR) tasks.

2.1 API

OpenSeq2Seq provides a top-level `run.py` script which takes a flexible Python configuration file specifying the model and execution mode (*train*, *eval*, *train_eval* or *infer*). The configuration file allows user to specify parts of the model (i.e. data layer, encoder, decoder and loss) and their configuration parameters. Since the configuration file is written in Python, it is possible to provide actual Python classes as parameters. This maximizes flexibility by enabling users to define their own implementations for various components (e.g. encoders-decoders or even a custom learning rate decay schedules) without modifying the toolkit source code.

3 Mixed Precision support

OpenSeq2Seq fully supports training with mixed precision using float16 data types to utilize the newest GPUs. When using float16 to train large state-of-the art models, it is sometimes necessary to apply certain algorithmic techniques and keep some outputs in float32 (hence, mixed precision) to achieve best results. For mixed precision training we follow an algorithmic recipe from (Micikevicius et al., 2017). At a high level it can be summarized as follows:

1. Maintain float32 master copy of weights for weights update while using the float16 weights for forward and back propagation

2. Apply loss scaling while computing gradients to prevent underflow during back-propagation

It is worth mentioning that both (1)-(2) are not always necessary. However, this method has proven to be robust across a variety of large set of complex models (Micikevicius et al., 2017).

Note that while having two copies of weights increase the memory consumption for weights, the total memory requirements for mixed precision is often *decreased* because activations, activation gradients, and other intermediate tensors can now be kept in float16. This is especially beneficial for models with a high degree of parameter sharing, such as recurrent models.

3.1 Mixed Precision Optimizer

Our implementation is different from the previous approach[1]: instead of a custom variable getter, we introduce a wrapper around standard TensorFlow optimizers. The model is created with float16 data type, so all variables and gradients are in float16 by default (except for the layers which are explicitly redefined as float32; for example data layers or operations on CPU). The wrapper then automatically converts float16 gradients to float32 and submits them to TensorFlow's optimizer, which updates the master copy of weights in float32. Updated float32 weights are converted back to float16 weigths, which are used by the model in the next forward-backward iteration. Figure 1 illustrates the `MixedPrecisionOptimizerWrapper` architecture.

3.2 Mixed Precision Regularizer

Training in mixed precision may need special care for regularization. Consider, for example, weight decay regularization when weights decay term $2\lambda * w$ is added to the gradients with respect to the loss $\frac{\partial L}{\partial w}$. Given that the weights are commonly initialized with small values, multiplying them with weight decay coefficient λ which is usually on the order of $[10^{-5}, 10^{-3}]$ can result in numerical underflow.

To overcome this problem we use the following approach. First, all regularizers should be defined during variable creation (a regularizer parameter in the `tf.get_variable` function or `tf.layers` objects). Second, the regularizer function should be wrapped with `mp_regularizer_wrapper` function which does two things. First, it adds variable with the user-provided regularization function to the TensorFlow collection. Second, it disables the underlying regularization function for float16 copy. The created collection will later be retrieved by `MixedPrecisionOptimizerWrapper` and the corresponding functions will be applied to the float32 copy of the weights ensuring that their gradients always stay in full precision. Since this regularization is not in the loss computation graph, we explicitly call `tf.gradients` and add the result to the gradients passed in the `compute_gradients` in the optimizer.

[1] http://docs.nvidia.com/deeplearning/sdk/mixed-precision-training/. Accessed: 2018-04-06.

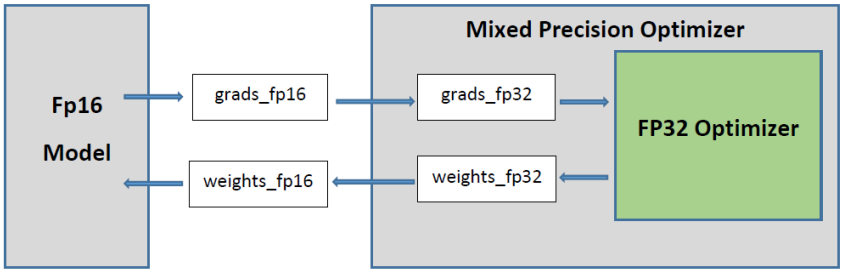

Figure 1: "Mixed precision" optimizer wrapper around any TensorFlow optimizer

3.3 Automatic Loss Scaling

The mixed precision training approach suggests that the user set a fixed *loss scale* hyper-parameter to adjust the dynamic range of backpropagation to match the dynamic range of float16 (Micikevicius et al., 2017). OpenSeq2Seq implements an *automatic loss scaling* so the user does not have to select the loss-scale value manually. The optimizer inspects the parameter gradients at each iteration and uses their values to select the loss scale for the *next* iteration.

4 Machine Translation and Automatic Speech Recognition

NMT and ASE are two modalities which currently have full implementation in OpenSeq2Seq.

4.1 NMT experiments

Neural Machine Translation (NMT) is naturally expressed in terms of encoder-decoder paradigm (Bahdanau et al., 2014; Wu et al., 2016; Vaswani et al., 2017). OpenSeq2Seq provides several encoder and decoder implementations for this task - RNN and non-RNN based ones with various types of attention. It has all the necessary blocks for GNMT-like (Wu et al., 2016) and Transformer (Vaswani et al., 2017) models. Also, these blocks can be easily mixed together.

For example, if the user wants to train GNMT-like (Wu et al., 2016) model he/she needs to construct a configuration file which uses `GNMTLikeEncoderWithEmbedding` as the encoder, and `RNNDecoderWithAttention` as the decoder for training and `BeamSearchRNNDecoderWithAttention` for inference. The precision mode (float32, float16 or mixed) as well as the number of GPUs and other parameters are also specified in the configuration file.

For training using mixed precision we do not change network topology or any of the hyper parameters. Figure 2 (A) demonstrates that training loss curves for GNMT-like model using float32 and mixed precision track each other very closely during training (the same is true for Transformer training). In our experiments, we used WMT 2016 English→German data set obtained by combining the Europarlv7, News Commentary v10, and Common Crawl corpora and resulting in roughly 4.5 million sentence pairs. Table 1 compares BLEU scores after training with float32 and mixed precision. These scores are computed using *multi-bleu.perl*[2] script from Moses against *newstest2013.tok.de* file.

Model	Iteration	Score
GNMT-like float32	340K	23.21 BLEU
GNMT-like mixed	340K	23.63 BLEU
Transformer float32	220K	25.2 BLEU
Transformer mixed	220K	25.4 BLEU
DS2-like float32	110K	4.59% WER
DS2-like mixed	110K	4.47% WER

Table 1: Evaluation scores after training using float32 and mixed precision. We used 2, 4 and 8 GPUs to train Transformer, GNMT and DeepSpeech2 models. All configs are available on OpenSeq2Seq's GitHub.

In our experiments, for Transformer and GNMT-like model, total GPU memory consumption is reduced to about 55% of what it was while using float32. We also observe performance boosts (around x1.8 for GNMT) which can vary depending on the batch size. The general rule of thumb is that bigger batch size yields better performance.

[2] https://github.com/moses-smt/mosesdecoder/blob/master/scripts/generic/multi-bleu.perl

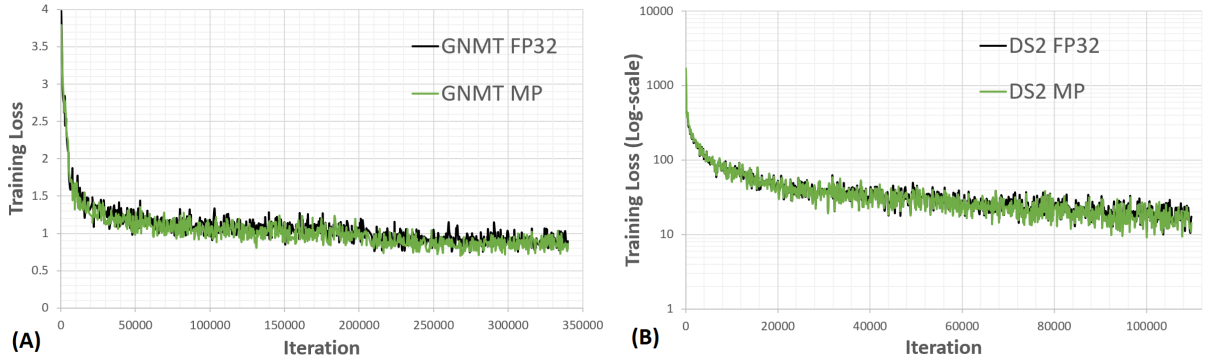

Figure 2: Training loss curves for: (A) GNMT-like model, and (B) DeepSpeech2-like model using float32 and mixed precision. For both models, float32 and mixed precision training very closely match each other.

4.2 ASR experiments

Many recent Automated Speech Recognition (ASR) models are built using explicit encoder-decoder topology (Battenberg et al., 2017; Prabhavalkar et al., 2017; Chiu et al., 2017). However, even for models without explicit encoder-decoder topology, it is easy to re-formulate them as such in our toolkit. For example, let's consider an encoder-decoder implementation of Deep Speech 2 (DS2) model (Amodei et al., 2016) in OpenSeq2Seq. DS2 consists of three convolutional layers, several bidirectional or unidirectional recurrent layers (with LSTMs or GRUs), an optional row convolutional layer, and a fully connected layer followed by a hidden layer which produces logits for the Connectionist Temporal Classification (CTC) loss (Graves et al., 2006). Logits represent a probability distribution over alphabet characters at each timestep. A beam search CTC decoder with language model rescorer is usually employed for producing the output characters sequence during inference. While DS2 doesn't contain explicit encoder and decoder (in seq2seq sense), we can split the model in the following fashion: convolutional, recurrent and fully connected layers are encapsulated in the `DeepSpeech2Encoder`, and logits' hidden layer together with the CTC decoder are encapsulated in the `FullyConnectedCTCDecoder`. The reason behind this split point is simple: it allows the encoder to output a custom-sized representation and it encourages encoder and decoder re-use. For example, the CTC decoder can be replaced with any decoder from text-to-text models.

In our experiments, we trained DeepSpeech2-like model on a "clean" and "other" subsets of LibriSpeech training dataset (Panayotov et al., 2015). Table 1 shows final Word Error Rates (WER)[3] on LibriSpeech "dev-clean" subset, obtained after training using float32 and mixed precision. Similarly to GNMT experiments, we did not change any of the hyper parameters when training in mixed precision. During training in mixed precision, we observed a total memory reduction to around 57% compared to float32 mode. Figure 2 (B) demonstrates that mixed precision has no effect on convergence behaviour.

5 Conclusion and future plans

Modern deep learning hardware is moving towards training with low precision. NVIDIA's Volta-based GPUs offer significant performance boost and reduced memory footprint while training using Tensor Cores (e.g. using float16)[4]. OpenSeq2Seq natively supports training using mixed precision and allows NLP and ASR researchers to increase their productivity. In our experiments, we see total memory reductions to 55%–57% of float32 mode for GNMT, Transformer and DeepSpeech2 models.

OpenSeq2Seq aims to offer a rich library of commonly used encoders and decoders. We plan to extend it with other modalities such as text-to-speech and image-to-text. Finally, we are working on providing more encoder and decoder choices for already supported tasks such as machine translation and speech recognition.

[3] With beam width = 2048 and language model provided by Mozilla: https://github.com/mozilla/DeepSpeech/tree/master/data/lm

[4] http://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf

Acknowledgments

We are grateful to Siddharth Bhatnagar and Luyang Chen for their work on previous version of the toolkit. We would like to thank Hao Wu, Ben Barsdell, Nathan Luehr, Jonah Alben, and Ujval Kapasi for their fruitful discussions.

References

Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. Tensorflow: A system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283.

Dario Amodei, Rishita Anubhai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Jingdong Chen, Mike Chrzanowski, Adam Coates, Greg Diamos, Erich Elsen, Jesse Engel, Linxi Fan, Christopher Fougner, Awni Y. Hannun, Billy Jun, Tony Han, Patrick LeGresley, Xiangang Li, Libby Lin, Sharan Narang, Andrew Y. Ng, Sherjil Ozair, Ryan Prenger, Sheng Qian, Jonathan Raiman, Sanjeev Satheesh, David Seetapun, Shubho Sengupta, Chong Wang, Yi Wang, Zhiqian Wang, Bo Xiao, Yan Xie, Dani Yogatama, Jun Zhan, and Zhenyao Zhu. 2016. Deep speech 2 : End-to-end speech recognition in english and mandarin. In *ICML*.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Eric Battenberg, Jitong Chen, Rewon Child, Adam Coates, Yashesh Gaur, Yi Li, Hairong Liu, Sanjeev Satheesh, David Seetapun, Anuroop Sriram, et al. 2017. Exploring neural transducers for end-to-end speech recognition. *arXiv preprint arXiv:1707.07413*.

Denny Britz, Anna Goldie, Thang Luong, and Quoc Le. 2017. Massive Exploration of Neural Machine Translation Architectures. *ArXiv e-prints*.

William Chan, Navdeep Jaitly, Quoc V Le, and Oriol Vinyals. 2015. Listen, attend and spell.

Chung-Cheng Chiu, Tara N Sainath, Yonghui Wu, Rohit Prabhavalkar, Patrick Nguyen, Zhifeng Chen, Anjuli Kannan, Ron J Weiss, Kanishka Rao, Katya Gonina, et al. 2017. State-of-the-art speech recognition with sequence-to-sequence models. *arXiv preprint arXiv:1712.01769*.

Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. 2006. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd international conference on Machine learning*, pages 369–376. ACM.

Guillaume Klein, Yoon Kim, Yuntian Deng, Jean Senellart, and Alexander M. Rush. 2017. Opennmt: Open-source toolkit for neural machine translation. In *Proc. ACL*.

Minh-Thang Luong, Eugene Brevdo, and Rui Zhao. 2017. Neural machine translation (seq2seq) tutorial. *https://github.com/tensorflow/nmt*.

Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaev, Ganesh Venkatesh, et al. 2017. Mixed precision training. *arXiv preprint arXiv:1710.03740*.

Vassil Panayotov, Guoguo Chen, Daniel Povey, and Sanjeev Khudanpur. 2015. Librispeech: an asr corpus based on public domain audio books. In *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*, pages 5206–5210. IEEE.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in pytorch.

Rohit Prabhavalkar, Kanishka Rao, Tara N Sainath, Bo Li, Leif Johnson, and Navdeep Jaitly. 2017. A comparison of sequence-to-sequence models for speech recognition. In *Proc. Interspeech*, pages 939–943.

Alexander Sergeev and Mike Del Balso. 2018. Horovod: fast and easy distributed deep learning in tensorflow. *arXiv preprint arXiv:1802.05799*.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 3104–3112. Curran Associates, Inc.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 6000–6010.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*.

Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. 2015. Show, attend and tell: Neural image caption generation with visual attention. In *International Conference on Machine Learning*, pages 2048–2057.

Dong Yu, Adam Eversole, Mike Seltzer, Kaisheng Yao, Zhiheng Huang, Brian Guenter, Oleksii Kuchaiev, Yu Zhang, Frank Seide, Huaming Wang, et al. 2014. An introduction to computational networks and the computational network toolkit. *Microsoft Technical Report MSR-TR-2014–112*.