The Sockeye 2 Neural Machine Translation Toolkit at AMTA 2020

Tobias Domhan domhant@amazon.com
Michael Denkowski mdenkows@amazon.com
David Vilar dvilar@amazon.com
Xing Niu xingniu@amazon.com
Felix Hieber fhieber@amazon.com
Amazon
Kenneth Heafield[*] translate@kheafield.com
Efficient Translation Limited

Abstract

We present Sockeye 2, a modernized and streamlined version of the Sockeye neural machine translation (NMT) toolkit. New features include a simplified code base through the use of MXNet's Gluon API, a focus on state of the art model architectures, distributed mixed precision training, and efficient CPU decoding with 8-bit quantization. These improvements result in faster training and inference, higher automatic metric scores, and a shorter path from research to production.

1 Introduction

Sockeye (Hieber et al., 2017) is a versatile toolkit for research in the fast-moving field of NMT. Since the initial release, it has been used in at least 25 scientific publications, including winning submissions to WMT evaluations (Schamper et al., 2018). Sockeye also powers Amazon Translate, showing industrial-strength performance in addition to the flexibility needed in academic environments. Moreover, we are excited to see that hardware manufacturers are contributing to optimizing MXNet (Chen et al., 2015) and Sockeye for speed. Intel has demonstrated large performance gains for Sockeye inference on Intel Skylake processors.[1] NVIDIA is working on significant performance improvements for Sockeye's Transformer (Vaswani et al., 2017) implementation through fused operators and an optimized beam search. This paper discusses Sockeye 2's streamlined Gluon implementation (§2), support for state of the art architectures and efficient decoding (§3), and improved model training (§4).

2 Gluon Implementation

Sockeye 2 is implemented using Gluon,[2] MXNet's latest and preferred API that combines the strengths of imperative and graph-based programming. Gluon provides a simple Python interface and uses eager execution by default, allowing developers to quickly prototype deep learning models and debug issues step by step. At runtime, Gluon can automatically "hybridize" models

[*] Work was done in an external advisory capacity.

[1] `https://www.intel.ai/amazing-inference-performance-with-intel-xeon-scalable-processors/#gs.wrgsji`

[2] `https://mxnet.apache.org/versions/1.6/api/python/docs/api/gluon/index.html`

	DE–EN		EN–DE		FI–EN		EN–FI	
Layers	BLEU	Latency	BLEU	Latency	BLEU	Latency	BLEU	Latency
6:6	35.5	602	37.9	791	22.2	575	20.5	808
10:10	35.4	970	37.8	1238	22.3	863	20.8	1258
20:2	34.8	293	37.6	357	23.2	257	20.9	368

Table 1: SacreBLEU scores (Post, 2018) and single-sentence latency in milliseconds on newstest2019 for models trained on WMT19 constrained data with varying numbers of encoder and decoder layers. Latency values are the 90th percentile of translation time when translating each sentence individually (no batching). We measure single sentence decoding latency on an EC2 c5.2xlarge instance with 4 CPU cores. We report the average over three independent training runs.

by converting them into computation graphs for maximum performance. Adopting this programming model significantly simplifies Sockeye 2's training and inference code, reducing the overall Python line count by 25%. Sockeye 2's hybridized transformer also improves training speed by 14% compared to Sockeye.

3 Focus on State of the Art Models

Due to the success of self-attentional models, we concentrate development of Sockeye 2 on the Transformer architecture (Vaswani et al., 2017). Our starting point is the "base" transformer with 6 encoder and decoder layers, model dimensionality of 512, and feed-forward layer size of 2048. An exploration of different encoder and decoder depths shows that deep encoders with shallow decoders are competitive in BLEU and significantly faster for decoding. Table 1 shows results with different numbers of encoder and decoder layers, denoted by x:y where x is the number of encoder layers and y the number of decoder layers. For WMT19 FI-EN and EN-FI benchmarks (Barrault et al., 2019), the 20:2 model outperforms both the 6:6 model and the 10:10 model in terms of BLEU. The 20:2 model also has roughly half the decoding latency of the 6:6 model and roughly one third the latency of the 10:10 model. The relative efficiency of encoder versus decoder layers can be attributed to (1) the ability to parallelize across input tokens, (2) attention to only input tokens, and (3) not needing to run beam search on the source side.

3.1 Source Factors

Sockeye supports source factors in the spirit of Sennrich and Haddow (2016), additional representations that are combined with word embeddings prior to the first encoder layer. In Sockeye 2, we improve source factor support by allowing different types of embedding combinations (concatenation, summation, or average), as well as weight sharing between source factor and word embeddings.

As an example application, we use source factors to represent input case. Variations in case pose a challenge for machine translation systems as different orthographic variations are considered to be independent by the translation model (e.g., "case" is different from "Case" and both are different from "CASE"). We address these variations by lowercasing the input and encoding the original case information as a source factor ("lowercase", "capitalized", "all uppercase" or "mixed"). We refer to this method as "SF-case". An alternative is to lowercase and include the original cased word itself as a source factor, which we refer to as "SF-word". As the original and lowercased versions of many words will be the same, it is useful to share the embeddings, a variant we refer to as "SF-word-share".

	DE – EN				EN – FI			
	Ori	lower	Cap	UPP	Ori	lower	Cap	UPP
Baseline (cased)	36.7	33.0	22.9	9.1	20.7	16.4	3.4	1.1
SF-case (concat)	36.8	34.7	24.2	26.6	20.7	18.2	9.3	7.0
SF-case (sum)	36.8	34.8	23.0	28.4	21.4	18.3	7.7	7.2
SF-word	36.5	33.6	23.1	9.2	20.9	17.0	4.4	1.4
SF-word-share	36.8	33.8	21.9	9.3	21.4	17.2	4.0	1.3

Table 2: Robustness results for several variants of representing case with source factors. Models are evaluated on transformed versions of newstest2019: **Ori**ginal case, **lower**cased, **Cap**italization of the first character of each word, and **UPP**ERCASED. Scores are case-insensitive SacreBLEU (Post, 2018).

To evaluate the robustness of these strategies, we modify test sets by either entirely lowercasing, entirely uppercasing, or capitalizing the first character of each word. We compare a baseline model that was trained on cased input (no source factors) against all "SF-*" methods. The factored models also use BPE type factors as introduced by Sennrich and Haddow (2016). Models use the 20:2 transformer architecture and training settings described in §3. Shown in Table 2, encoding case information with source factors is an effective way to improve robustness against case variation with the two versions of "SF-case" performing best.

3.2 Quantization for Inference

Sockeye 2 now supports 8-bit quantized matrix multiplication (Quinn and Ballesteros, 2018) on CPUs based on the `intgemm` library.[3] By scaling values such that 127 corresponds to the maximum absolute value found in a tensor, matrix multiplication can be conducted with 8-bit integer representations in place of the default 32-bit floating-point representations without significant degradation of overall model accuracy. Parameters can either be quantized offline and stored in a smaller model file or quantized on the fly at loading time. Activations are quantized on the fly while other operators that consume far less runtime remain as 32-bit floats.

Latency-sensitive applications typically run with batch size 1 and small beam sizes, leaving little opportunity for batch parallelism. Instead, matrix multiplication parallelizes over outputs of a layer. To reduce latency, matrix multiplication and quantization are both parallelized with OpenMP.[4] Layer outputs can be computed independently and the layer size is typically much larger than the batch size. Parallelizing over layer inputs would require summing across threads.

Shown in Table 3, quantization significantly reduces non-batched decoding times with minimal effect on BLEU scores. Improvement is most pronounced when running on a single CPU core while models using up to 4 cores still see a significant benefit.[5]

4 Training Improvements

Sockeye 2 significantly accelerates training with Horovod[6] integration (Sergeev and Balso, 2018) and MXNet's automatic mixed precision (AMP). Horovod extends synchronous training to any number of GPUs (including across nodes) while AMP automatically detects and converts parts of the model that can run in FP16 mode without loss of quality. These methods

[3] `https://github.com/kpu/intgemm`

[4] `https://www.openmprtl.org`

[5] For 1 and 2 cores, we set the number of OpenMP threads to 1 and 2 respectively. For 4 cores, we set the number of OpenMP threads to 3 for best interaction with MXNet's own parallelization over operators.

[6] `https://github.com/horovod/horovod`

		6:6 Layers			20:2 Layers		
	CPUs	Time (s)	Tok/Sec	BLEU	Time (s)	Tok/Sec	BLEU
Baseline (fp32)	1	1260.8	33.3	22.1	585.0	71.8	23.0
	2	841.6	49.9	22.1	404.7	103.8	23.0
	4	575.8	73.0	22.1	283.2	148.3	23.0
Quantized (int8)	1	511.6	82.1	22.0	285.7	147.0	22.8
	2	435.9	96.4	22.0	242.3	173.4	22.8
	4	334.3	125.7	22.0	173.0	242.9	22.8

Table 3: CPU decoding times and SacreBLEU (Post, 2018) scores for FI-EN newstest2019 with and without 8-bit quantization for both standard (6:6 layer) and deep encoder (20:2 layer) transformer models as described in §3. Models use a vocabulary selection shortlist of 200 items (Devlin, 2017) and translate one sentence at a time (batch size of 1). Benchmarks are run on an EC2 c5.12xlarge instance (Cascade Lake processor) and limited to using 1, 2, or 4 CPU cores.

	DE–EN		EN–FI	
	BLEU	Time	BLEU	Time
Ott et al. (2018)	34.7	30h	20.1	14h
Plateau-Reduce	**34.9**	**28h**	**20.7**	**12h**

Table 4: SacreBLEU (Post, 2018) scores (newstest2019) and training times (8 NVIDIA V100 GPUs) for a 20 encoder 2 decoder layer transformer using the training setup described by Ott et al. (2018) and plateau-reduce, both implemented in Sockeye 2.

also require additional computation per update (synchronizing data across distributed GPUs and checking reduced precision operations for overflow). This overhead can be amortized by significantly increasing the effective batch size; gradients are aggregated per-GPU for several batches, then combined and checked for overflow for a single parameter update. In practice, scaling the effective batch size by N, the learning rate by $\sqrt{N}$ (Krizhevsky, 2014), and leaving other hyper parameters unchanged works well for batches of up to 260K tokens.

Sockeye also provides a data-driven alternative to the popular "inverse square root" learning schedule used by Vaswani et al. (2017) and Ott et al. (2018). Termed "plateau-reduce", this scheduler keeps the same learning rate until validation perplexity does not increase for several checkpoints, at which time it reduces the learning rate and rewinds all model and optimizer parameters to the best previous point. Training concludes when validation perplexity reaches an extended plateau. In a WMT19 benchmark (Barrault et al., 2019), plateau-reduce training produces stronger models in slightly less time than the setup described by Ott et al. (2018). The results are presented in Table 4 where all values are averages over 3 independent training runs with different random initializations and all models train until validation perplexity reaches a plateau.

The relevant hyper parameters for Sockeye 2's large batch training are an effective batch size of 262,144 tokens, a learning rate of 0.00113 with 2000 warmup steps and a reduce rate of 0.9, a checkpoint interval of 125 steps, and learning rate reduction after 8 checkpoints without improvement. After an extended plateau of 60 checkpoints, the 8 checkpoints with the lowest validation perplexity are averaged to produce the final model parameters. While Horovod enables scaling to any number of GPUs, we find that training on 8 GPUs on a single node still delivers the best value when considering both speed and cost.

5 Licensing and availability

Sockeye 2 is available[7] under the Apache 2.0 license. It includes Docker builds to easily run training or inference with all of the latest features on any supported platform.

6 Conclusion

Sockeye 2 provides out-of-the-box support for quickly training strong Transformer models for research or production. Extensive configuration options and the simplified Gluon code base enable rapid development and experimentation. As an open source project, we invite the community to contribute their ideas to Sockeye 2 and hope that the new programming model and various performance improvements enable others to conduct effective and successful research.

References

Barrault, L., Bojar, O., Costa-jussà, M. R., Federmann, C., Fishel, M., Graham, Y., Haddow, B., Huck, M., Koehn, P., Malmasi, S., Monz, C., Müller, M., Pal, S., Post, M., and Zampieri, M. (2019). Findings of the 2019 conference on machine translation (WMT19). In *Procs. of the Fourth Conference on Machine Translation (Vol. 2: Shared Task Papers)*, pages 1–61, Florence, Italy.

Chen, T., Li, M., Li, Y., Lin, M., Wang, N., Wang, M., Xiao, T., Xu, B., Zhang, C., and Zhang, Z. (2015). Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *arXiv preprint arXiv:1512.01274*.

Devlin, J. (2017). Sharp models on dull hardware: Fast and accurate neural machine translation decoding on the cpu. *ArXiv e-prints*, abs/1705.01991.

Hieber, F., Domhan, T., Denkowski, M., Vilar, D., Sokolov, A., Clifton, A., and Post, M. (2017). Sockeye: A toolkit for neural machine translation. *ArXiv e-prints*, abs/1712.05690.

Krizhevsky, A. (2014). One weird trick for parallelizing convolutional neural networks. *arXiv preprint arXiv:1404.5997*.

Ott, M., Edunov, S., Grangier, D., and Auli, M. (2018). Scaling neural machine translation. In *Procs. of the Third Conference on Machine Translation, Vol. 1: Research Papers*, pages 1–9, Belgium, Brussels.

Post, M. (2018). A call for clarity in reporting bleu scores. In *Procs. of the Third Conference on Machine Translation, Vol. 1: Research Papers*, pages 186–191, Belgium, Brussels.

Quinn, J. and Ballesteros, M. (2018). Pieces of eight: 8-bit neural machine translation. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 3 (Industry Papers)*, pages 114–120, New Orleans - Louisiana. Association for Computational Linguistics.

Schamper, J., Rosendahl, J., Bahar, P., Kim, Y., Nix, A., and Ney, H. (2018). The RWTH Aachen University supervised machine translation systems for WMT 2018. In *Procs. of the Third Conference on Machine Translation, Vol. 2: Shared Task Papers*, pages 500–507, Belgium, Brussels.

Sennrich, R. and Haddow, B. (2016). Linguistic input features improve neural machine translation. In *Proceedings of the First Conference on Machine Translation: Volume 1, Research Papers*, pages 83–91, Berlin, Germany. Association for Computational Linguistics.

Sergeev, A. and Balso, M. D. (2018). Horovod: fast and easy distributed deep learning in tensorflow. *CoRR*, abs/1802.05799.

[7] `https://github.com/awslabs/sockeye`

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.