Accelerating Large Scale Deep Learning Inference through DeepCPU at Microsoft

Minjia Zhang, Samyam Rajbandari, Wenhan Wang, Elton Zheng, Olatunji Ruwase, Jeff Rasley, Jason Li, Junhua Wang, and Yuxiong He, ***Microsoft AI and Research***

https://www.usenix.org/conference/opml19/presentation/zhang-minjia

This paper is included in the Proceedings of the 2019 USENIX Conference on Operational Machine Learning (OpML '19).

May 20, 2019 • Santa Clara, CA, USA

ISBN 978-1-939133-00-7

Open access to the Proceedings of the 2019 USENIX Conference on Operational Machine Learning is sponsored by USENIX.

Accelerating Large Scale Deep Learning Inference through DeepCPU at Microsoft

Minjia Zhang, Samyam Rajbandari, Wenhan Wang, Elton Zheng,
Olatunji Ruwase, Jeff Rasley, Jason Li, Junhua Wang, Yuxiong He
Microsoft AI & Research
{minjiaz,samyamr, wenhanw, Elton.Zheng, olruwase, Jeff.Rasley, jasol, junhuaw, yuxhe}@microsoft.com

Abstract

The application of deep learning models presents significant improvement to many Microsoft services and products. In this paper, we introduce our experience and methodology of developing and applying the DeepCPU library for serving DL models in production at large scale with remarkable latency improvement and infrastructure cost reduction. We describe two ways to use the library, through customized optimization or framework integration, targeting different scenarios.

1 Introduction

Deep learning (DL) sits at the core of many essential products and services at Microsoft, such as web question and answering, web relevance ranking, advertising, language modeling, text translation, and conversational bot [2, 3, 4, 5, 7, 8]. Many of these services are deployed in large scale, supporting millions of users and billions of requests.

Such large scale DL inference faces *threefold* challenges to deploy a trained DL model at production. First, users expect to receive an inference result with low latency. The serving system needs to provide adequate quality of service, expressed as latency SLA (service level agreement), which is often a few milliseconds [11]. In practice, DL models are computationally expensive, incurring long latency, e.g., ranging from hundreds of milliseconds to seconds, that blocks their deployment [12, 23]. Second, when the volume of requests exceeds the capacity of a single server, the DL service must scale horizontally. An efficient serving system reduces the required replications and save thousands of machines and millions of cost. Finally, these constraints come together with restriction on the deployment infrastructure. In particular, it is strongly preferable to use existing commodity hardware, one main reason being the easier maintenance of the infrastructure and the agility of deployment.

To tackle these challenges, we foremost rely on a large amount of CPUs for serving DL models and adopt a co-development methodology called *SLT (scenario, library, and technique)* to make the best use of the CPU resource for business critical scenarios while accelerating the iteration cycle of deployment and optimization. In this paper, we present the SLT methodology and how it leads to DeepCPU, a DL inference library, which is deployed in production for many services on thousands of servers, and is tailored for DL scenarios with large number of users. We show two ways of applying DeepCPU, either through customized end-to-end optimized DL serving solution or low-level interface integration into frameworks such as TensorFlow [9] and ONNX [6]. Our evaluation on production models demonstrates the ability of the DeepCPU library that addresses latency SLA violation problem on a single server and also improves the throughput so that the DL service scales horizontally.

Scenarios	Services	Major components
Deep feature	Encoder model	GRU, Conv
	Embedding model	Stacked Bidir RNN, MLP, Attention
Web Q&A	MRC model A	Bidir RNN, Attention
	MRC model B	Bidir LSTM, Stacked LSTM, Conv, MLP, Attention
	MRC model C	Bidir GRU, Conv, MLP, Attention
Similarity ranking	Ranking model A	RNN encoder/decoder, Attention
	Ranking model B	GRU, Conv, MaxPool, Scoring
Query processing	Query rewriting	RNN encoder/decoder
	Query tagging	Stacked RNN

Table 1: DL scenarios and corresponding models.

2 Scenario, Library, and Technique (SLT)

This section highlights the SLT methodology. Section 2.1 describes DL inference scenarios that are of interest in our production. Section 2.2 introduces what DeepCPU library is. Section 2.3 shows our performance optimization techniques.

2.1 Major Deep Learning Scenarios

We start the SLT methodology with a bird's-eye view of some major Microsoft scenarios that leverage DL models from the standpoint of latency SLA and resource consumption. Table 1 shows some of the scenarios, services, and model components.

Deep feature uses a DL model to encode entities (e.g., text) into deep descriptors (i.e., vectors). The generated vectors are used for semantic understanding of downstream models.

Web Q&A addresses web question-and-answering scenario. It uses a machine reading comprehension model to generate a high quality answer based on the question in a query.

Similarity ranking reranks the top-N text passages for each query based on their semantic similarity to the query.

Query rewriting performs sequence-to-sequence rewriting to map a query to some other query (well corrected, altered, paraphrased) at runtime and uses this query to surface more and better documents for the query.

Query tagging identifies entities in the query to enable more precise matching with documents.

These are just a few examples. There are many more services that leverage DL models in various forms. These services often face challenges from latency, cost, or both. For example, for MRC models, latency is often a big challenge. MRC model A has serving latency of 200ms using TensorFlow [9] but requires to meet 10ms latency SLA for shipping.

DL services	Original Latency	Latency Target	Optimized Latency	Latency Reduction	Throughput Improvement
Encoder model	∼29ms	10ms	5.4ms	5X	5X
MRC model A	∼100ms	10ms	9ms	>10X	>10X
MRC model B	∼107ms	10ms	4.1ms	>20X	>50X
MRC model C	∼45ms for batch size 1	10ms	<8.5ms for batch size 20	11X	>100X
Ranking model A	10∼12ms for batch size 1	6ms	<6ms for batch size 33	>6X	>30X
Ranking model B	10ms for batch size 1	6ms	<5ms for batch size 150	>10X	>100X
Query rewriting	51ms	5ms	4ms	>10X	>3X
Query tagging	9∼16ms	3ms	0.95ms	10X	>10X
NMT model	29ms	10ms	5.8ms	5X	5X

Table 2: Optimization results with and without DeepCPU on production models.

For similarity ranking models, cost is often a big concern. Ranking model A takes 10ms to serve a query with batch size 1 on a single server, whereas the latency SLA is 5ms for batch size 150. This is not scalable because even a fan-out solution requires thousands of machines to serve the large volume of request traffic.

2.2 Highly Reusable Library

Table 1 also shows the DL components each model has. Those components are divided into three categories.

RNN family includes GRU/LSTM cell and sequence, unidirectional/bidirectional RNN, and stacked RNNs [10, 13].

Fundamental building blocks and common DL layers includes matrix-multiply kernels, high-way network [20], max pooling layer [16], Conv layer [15], MLP layer [18], etc.

DL layers for machine reading comprehension and conversation models includes variety of attention layers [14, 19], seq2seq decoding with beam search [21], etc.

We build DeepCPU, a library of these components as building blocks with customized optimization. We find that these components are highly reusable and allow faster implementation and decreased development cost to support new scenarios. As an example, it takes < 200 lines of C++ code for running a Seq2Seq model end-to-end with the library.

2.3 Performance Optimization Techniques

Not only we support the library, but we also offer optimization techniques to optimize different components. We perform three large categories of optimizations:

Intra-op optimizations. We provide i) more efficient matrix computation by combining Intel MKL [1] with customized cache-aware kernel computation to handle, large matrix computation, as well as small or tall-and-skinny matrix-multiplication. ii) optimized common activation functions using continued fraction expansion [22], efficient parallelization, and SIMD vectorization.

Inter-op optimizations. We perform operation fusion which fuses point-wise operation to avoid multiple scans of data and reduced data movement overhead.

Parallelism, scheduling, and affinity. The parallelism, load balancing, and scheduling order are also critical to the performance of DL optimization on multicore CPU. Existing frameworks such as TensorFlow are designed to handle generic DAG, which can lead to suboptimal parallelism decisions and cannot control per-op parallelism, while we consider the characteristics of the workload and perform global optimization by looking at model structure. We also pin application threads to physical cores and make DL computation NUMA-aware and socket-aware to avoid expensive context switching and cross-socket communication overhead.

3 How is DeepCPU Utilized?

DeepCPU is currently released as C++ SDK to first party users. There are two approaches to use the library.

Customized optimization. This approach requires rewriting the model runtime using the DeepCPU library. After then we tune the performance such as thread settings, targeting at obtaining the ultimately optimized performance, because at large scale, every possible bit of hardware optimization space leads to major improvements. This approach requires interaction with the model developer and requires some development efforts if the model changes drastically. To achieve improved performance with less development work, we also integrate DeepCPU into existing DL frameworks.

Framework integration. We replace frequently used and costly operators, such as LSTM, GRU, Conv2D, Attention, with DeepCPU 's high-performance implementations in TensorFlow runtime. This approach targets framework users directly, and it allows users to use existing frameworks to develop models while taking only a minimal amount of work to switch the operators to take the advantage of DeepCPU. Meanwhile, we are closely working with ONNX team to power ONNX runtime [6] with DeepCPU technology, which allows frameworks that support ONNX IR, such as PyTorch [17], to also benefit from DeepCPU.

For new scenarios and models, we often encourage to try the framework integration approach first, which allows fast deployment if that already gives satisfying performance results (e.g., meeting latency SLA). Otherwise, we apply customized optimization to further boost the performance.

4 Evaluation results

Table 2 shows a list of models we have optimized with DeepCPU, with both latency and throughput improvement in comparison with TensorFlow on a server with two 2.20 GHz Intel Xeon E5-2650 V4 processors, each of which has 12-core with 128GB RAM. Overall, we see 5–20 times latency improvement, which helps to change the model status from non–shippable to shippable. Meanwhile, we have achieved up to 100 times throughput improvement and cost reduction. These models have been running in production for the last two years on thousands of servers.

References

[1] Intel(R) Math Kernel Library. https://software.intel.com/en-us/mkl.

[2] Internet-Scale Deep Learning for Bing Image Search. https://blogs.bing.com/search-quality-insights/May-2018/Internet-Scale-Deep-Learning-for-Bing-Image-Search. Accessed: 27-January-2019.

[3] Machine Reading. https://www.ailab.microsoft.com/experiments/ef90706b-e822-4686-bbc4-94fd0bca5fc5. Accessed: 27-January-2019.

[4] Machine Reading at Scale – Transfer Learning for Large Text Corpuses. https://blogs.technet.microsoft.com/machinelearning/2018/10/17/machine-reading-at-scale-transfer-learning-for-large-text-corpuses/. Accessed: 27-January-2019.

[5] Microsoft is teaching systems to read, answer and even ask questions. https://blogs.microsoft.com/ai/microsoft-is-teaching-systems-to-read-answer-and-even-ask-questions/. Accessed: 27-January-2019.

[6] Open neural network exchange format (ONNX). https://github.com/onnx/onnx. Accessed: 27-January-2019.

[7] Towards More Intelligent Search: Deep Learning for Query Semantics. https://blogs.bing.com/search-quality-insights/May-2018/Towards-More-Intelligent-Search-Deep-Learning-for-Query-Semantics. Accessed: 27-January-2019.

[8] What's New in Deep Learning Research: Microsoft Wants Machines to Understand What They Read. https://medium.com/@jrodthoughts/whats-new-in-deep-learning-research-microsoft-wants-machines-to-understand-what-they-read-ebe61e1853a5. Accessed: 27-January-2019.

[9] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: A System for Large-scale Machine Learning. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, OSDI '16, pages 265–283, 2016.

[10] Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. arXiv preprint arXiv:1412.3555, 2014.

[11] Tobias Flach, Nandita Dukkipati, Andreas Terzis, Barath Raghavan, Neal Cardwell, Yuchung Cheng, Ankur Jain, Shuai Hao, Ethan Katz-Bassett, and Ramesh Govindan. Reducing Web Latency: The Virtue of Gentle Aggression. In *Proceedings of the ACM Conference of the Special Interest Group on Data Communication*, SIGCOMM '13, pages 159–170, 2013.

[12] Pin Gao, Lingfan Yu, Yongwei Wu, and Jinyang Li. Low Latency RNN Inference with Cellular Batching. In *Proceedings of the Thirteenth EuroSys Conference*, EuroSys '18, pages 31:1–31:15, 2018.

[13] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 1997.

[14] Rudolf Kadlec, Martin Schmid, Ondrej Bajgar, and Jan Kleindienst. Text Understanding with the Attention Sum Reader Network. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL '16*, 2016.

[15] Yoon Kim. Convolutional Neural Networks for Sentence Classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014*, pages 1746–1751, 2014.

[16] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012*, NIPS '12, pages 1106–1114, 2012.

[17] Adam Paszke, Sam Gross, Soumith Chintala, and Gregory Chanan. PyTorch: Tensors and dynamic neural networks in Python with strong GPU acceleration.(2017). *https://github.com/pytorch/pytorch*, 2017.

[18] Dennis W Ruck, Steven K Rogers, Matthew Kabrisky, Mark E Oxley, and Bruce W Suter. The multilayer perceptron as an approximation to a Bayes optimal discriminant function. *IEEE Transactions on Neural Networks*, 1(4):296–298, 1990.

[19] Min Joon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional Attention Flow for Machine Comprehension. arXiv preprint arXiv:1611.01603, 2016.

[20] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway Networks. arXiv preprint arXiv:1505.00387, 2015.

[21] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to Sequence Learning with Neural Networks. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014*, NIPS '14, pages 3104–3112, 2014.

[22] AJ Van der Poorten. Continued fraction expansions of values of the exponential function and related fun with continued fractions. *Nieuw Archief voor Wiskunde*, 14:221–230, 1996.

[23] Minjia Zhang, Samyam Rajbhandari, Wenhan Wang, and Yuxiong He. DeepCPU: Serving RNN-based Deep Learning Models 10x Faster. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, pages 951–965, Boston, MA, 2018. USENIX Association.