Exploring GPU Architectural Optimizations for RNNs

Suchita Pati
Computer Sciences Department
University of Wisconsin-Madison
spati@cs.wisc.edu

Abstract

GPUs have been widely adopted for the training of Deep Learning applications employing Convolutional Neural Networks (CNNs). However, Recurrent Neural Network (RNN) based applications, such as those for Natural Language Processing and Speech and Text Recognition, are becoming increasingly popular but have received significantly less attention. RNNs possess a different set of characteristics than CNNs, such as serialization between time-steps and the need to remember recent events, and are unable to take advantage of many of the common GPU hardware and software optimizations for CNNs. To overcome this, we propose holistic changes to the design of GPUs to optimize them for RNNs.

1 Introduction

Deep Learning has become immensely popular in the last decade, dominating fields such as robotics, virtual reality and autonomous vehicles. Given this, research on hardware for deep neural networks (DNNs) has also become imperative leading to the proposal of several accelerators [7–9, 15, 18, 19, 31, 34, 38]. However, more general-purpose GPUs are also widely used for running DNNs since they are more flexible and better able to adapt to new algorithms than accelerators, which quickly become obsolete due to rapidly evolving algorithms. Therefore, GPUs have been extensively used for both CNN training and inference [28–30].

GPUs have also been used for RNNs, although RNNs are less well studied than CNNs because CNNs are considered more widely used in practice [12]. However, RNNs are also an important application that systems must optimize for [17, 21, 24]. For example, Facebook runs 300 trillion ML inferences a day, most of which are RNNs [16], and Google found that 29% of the Tensor Processing Unit (TPU) workloads are RNNs (in comparison, only 5% are CNNs) [24]. RNNs present a unique challenge for GPUs, since each RNN job often uses kernels with few threads that cannot fully utilize the GPU [13]. Consequently, single RNN jobs cannot benefit from increasing the number of computational units.

Moreover, batching RNN jobs to increase hardware utilization is impractical for RNN inference, since many of the application domains where RNNs are widely used, such as speech recognition, have real-time constraints [17, 24, 43]. For example, data center data shows that an RNN inference job must be completed in 7-10 ms [24, 43]. As a result, RNN performance significantly lags that of CNNs across a wide variety of platforms [35, 36]. We further discuss RNNs and their unique processing model in Section 2.

YArch '19, Feb 2019, USA

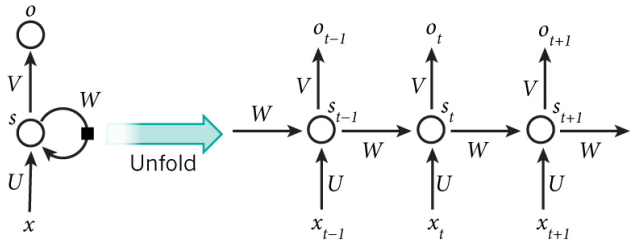

Figure 1. RNN structure and the unfolding in time of the computations involved in its forward pass [4].

The fundamental differences between RNNs and CNNs necessitate a different set of GPU optimizations that have not been previously explored. As discussed in Section 5, prior work on RNNs has primarily focused on software solutions such as efficient strategies for mapping the RNNs on to the GPUs [14, 44]. In contrast, we propose to holistically rethink the GPU architecture – including the microarchitecture, memory system, and coherence protocol, to optimize GPUs for RNNs.

2 Background and Challenges

An example RNN is shown in Figure 1. In a single hidden layer recurrent network, given that x_t is the input at time t and h_{t-1} is the recurrent layer activation matrix at time t-1, then the recurrent layer activations are calculated as follows:

$$h_t = \mathrm{f}\,(\mathrm{W}\,x_t + \mathrm{U}\,h_{t-1} + \mathrm{b}\,)$$

where W is the input-hidden weight matrix and U is the recurrent weight matrix and b is a bias term.

RNNs contain loops which allow them to capture and remember information across multiple iterations (or timesteps). These loops can be unrolled, as shown on the right of Figure 1. Each of the unrolled iterations passes information to the next iteration. This sequential dependency between iterations **limits parallelism**. Although input batching is one of the optimizations that has been adopted to overcome this issue for CNNs, this is not feasible for RNN (inference) due to strict latency requirements [24, 43].

The number of times the loop is unrolled, N, represents the sequence length of the RNN. Unrolling the RNN also allows it to be formulated as a series of matrix operations using optimized matrix multiplication routines like General Matrix Multiply (GEMM). The hidden state vector (the memory) is calculated by looking at the previous hidden state h_{t-1} and

the input h_t at the current step. In theory, RNNs could be unrolled an infinite number of times, which would be equivalent to remembering everything that has happened previously. In practice, the **large memory footprint and bandwidth limitations** of RNNs limit them to only remembering the most recent events. Despite this practical limitation, RNNs are well suited for domains where prior events persist and influence subsequent ones [2, 5, 6, 20, 40, 42].

3 Proposal

To overcome the challenges mentioned in Section 2, we propose several changes to the microarchitecture, memory system and coherence protocol of GPUs. The key insight underlying these innovations is that RNNs have producer-consumer parallelism that can be exploited at multiple levels to improve performance and energy efficiency.

3.1 Microarchitecture

Currently, RNNs must wait for all activation elements to be computed before the result can be sent to the next timestep. This incurs unnecessary overhead because some of the activations have been computed much earlier than other. By sending a partially computed activation element or activation matrix from the current timestep to carry out computations of the next timestep in parallel, we can exploit more parallelism at the microarchitectural level. To do this, we will map each timestep's computations to a different Streaming Multiprocessor (SM) via an intelligent scheduling mechanism. Thus, one SM will be a producer of activations for the next timestep, which is consumed by a different SM. Moreover, we will also distribute a timestep's computations across multiple SMs to improve parallelism, similar to prior work [14]. To ensure this scheduling is only used when desired, we will add a programmer-controlled flag.

3.2 Memory System

Since the activations are produced and consumed by different (sets of) SMs, memory system optimizations are also required.

Reduce Redundant Storage: We propose using the L1 cache for storing updated activations. Although prior work stores activations in shared memory [14], this requires multiple activation copies per SM whereas we only require one copy. By storing fewer copies per SM, we enable larger recurrent layer sizes. However, using caches requires TLB accesses which may increase the access latency but it would be small compared to repeatedly loading activations from global memory.

Reduce Memory Bandwidth: Storing the activations in caches significantly reduces the memory bandwidth requirements. However, caches pose an additional challenge since the activations may be evicted. Therefore, an additional mechanism is required to retain all the activations in the cache. To do this efficiently, we will build on prior work on locking cache ways [11], or making the shared memory globally addressable and coherent [27].

Reduce Communication Overhead: Although storing the activations in shared memory [14] avoids some of the overheads of caches, it also has its own set of issues because the shared memory must copy the data back and forth from the global memory to propagate updates. This incurs significant latency overhead as it requires updates to the global memory at the producer SM and loading of the updated activations into the shared memory of the consumer SM at every timestep. Thus, we will optimize the GPU coherence protocol [41] to propagate updated activations directly to the consumers' L1 cache.

4 Evaluation

There has been a significant effort towards building infrastructure for accurately simulating state-of-the-art GPU architectures in GPGPU-Sim (such as NVIDIA's Pascal and Volta GPUs [23, 25], including tensor cores [39]) with support for executing Deep Learning benchmarks that use cuBLAS, cuDNN, PyTorch, and TensorFlow [33]. Using the updated GPGPU-Sim infrastructure will allow us to rapidly prototype our ideas, while being confident that the changes are reflective of real hardware. As a first step, we have been augmenting GPGPU-Sim to simulate RNNs from DeepBench [35, 36], which has both training and inference implementations for the three most popular variants of RNN algorithms – Vanilla, GRU [10] and LSTM [22]. With slight modifications, we have been able to execute them with various batch sizes, number of hidden layer units, and timesteps, which will help us evaluate our ideas for different RNNs. In addition to this, we plan to compare against end-to-end solutions such as DeepSpeech2 [1], PRNN [14], and SRU [32].

5 Related Work

Recently, there has been an increasing amount of work on optimizing RNNs, especially at the software level. Research on matrix multiplication level optimizations has resulted in optimized cuDNN implementations that concatenate multiple inputs as a single large matrix through batching or concatenation of weight matrices for networks with wide hidden layers [**?**] for GPUs. Since our applications are written using cuDNN, our design is built on top of these optimizations. A similar effort occurred on the CPU side with the DeepCPU library [43]. However, CPUs do not provide the extent of parallelism we would require to implement our design.

GPUs have also been recently equipped with dedicated and specialized units for matrix multiplications (tensor cores [3]) which can significantly boost performance as RNNs gain considerably from reduced precision [14]. Since our infrastructure models current NVIDIA GPUs and has support for tensor cores, this is another feature that our work builds upon.

Most prior work on optimizing RNNs for GPUs focuses on modifying the network architecture, pruning network parameters and creating efficient mapping of the network on GPUs [14, 26, 32, 37, 45]. For example, Persistent RNNs [14]

retain the recurrent weight matrix within SMs over all timesteps. Although this approach works well for some RNNs, it also requires heavyweight synchronization. In comparison, our approach is more general.

References

[1] 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, Tony Han, Awni Y. Hannun, Billy Jun, Patrick LeGresley, Libby Lin, Sharan Narang, Andrew Y. Ng, Sherjil Ozair, Ryan Prenger, Jonathan Raiman, Sanjeev Satheesh, David Seetapun, Shubho Sengupta, Yi Wang, Zhiqian Wang, Chong Wang, Bo Xiao, Dani Yogatama, Jun Zhan, and Zhenyao Zhu. 2015. Deep Speech 2: End-to-End Speech Recognition in English and Mandarin. *CoRR* abs/1512.02595 (2015). arXiv:1512.02595 http://arxiv.org/abs/1512.02595

[2] 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 *Proceedings of the 33nd International Conference on Machine Learning (ICML)*. 173–182.

[3] Jeremy Appleyard and Scott Yokim. 2017. Programming Tensor Cores in CUDA 9. https://devblogs.nvidia.com/programming-tensor-cores-cuda-9/. (2017).

[4] Denny Britz. 2015. Recurrent Neural Networks Tutorial, Part 1 âĂŞ Introduction to RNNs. http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/. (2015).

[5] Denny Britz, Anna Goldie, Minh-Thang Luong, and Quoc Le. 2017. Massive Exploration of Neural Machine Translation Architectures. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. 1442–1451.

[6] William Chan and Ian Lane. 2015. Deep Recurrent Neural Networks for Acoustic Modelling. *CoRR* abs/1504.01482 (2015). arXiv:1504.01482 http://arxiv.org/abs/1504.01482

[7] Andre Xian Ming Chang, Berin Martini, and Eugenio Culurciello. 2015. Recurrent Neural Networks Hardware Implementation on FPGA. *CoRR* abs/1511.05552 (2015). arXiv:1511.05552 http://arxiv.org/abs/1511.05552

[8] Tianshi Chen, Zidong Du, Ninghui Sun, Jia Wang, Chengyong Wu, Yunji Chen, and Olivier Temam. 2014. Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. *ACM Sigplan Notices* 49, 4 (2014), 269–284.

[9] Yunji Chen, Tao Luo, Shaoli Liu, Shijin Zhang, Liqiang He, Jia Wang, Ling Li, Tianshi Chen, Zhiwei Xu, Ninghui Sun, et al. 2014. Dadiannao: A machine-learning supercomputer. In *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, 609–622.

[10] Kyunghyun Cho, Bart van Merriënboer, ÇaÄ§lar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, Doha, Qatar, 1724–1734. http://www.aclweb.org/anthology/D14-1179

[11] Henry Cook, Krste Asanovic, and David A Patterson. 2009. Virtual local stores: Enabling software-managed memory hierarchies in mainstream computing environments. *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-131* (2009).

[12] Eugenio Culurciello. 2018. The Fall of RNN / LSTM. https://towardsdatascience.com/the-fall-of-rnn-lstm-2d1594c74ce0. (2018).

[13] Eugenio Culurciello, Aliasger Zaidy, and Vinayak Gokhale. 2017. Computation and Memory Bandwidth in Deep Neural Networks. https://medium.com/@culurciello/computation-and-memory-bandwidth-in-deep-neural-networks-16cbac63ebd5. (2017).

[14] Greg Diamos, Shubho Sengupta, Bryan Catanzaro, Mike Chrzanowski, Adam Coates, Erich Elsen, Jesse Engel, Awni Y. Hannun, and Sanjeev Satheesh. 2016. Persistent RNNs: Stashing Recurrent Weights On-Chip. In *Proceedings of the 33nd International Conference on Machine Learning (ICML)*. 2024–2033. http://jmlr.org/proceedings/papers/v48/diamos16.html

[15] Zidong Du, Robert Fasthuber, Tianshi Chen, Paolo Ienne, Ling Li, Tao Luo, Xiaobing Feng, Yunji Chen, and Olivier Temam. 2015. ShiDianNao: Shifting vision processing closer to the sensor. In *ACM SIGARCH Computer Architecture News*, Vol. 43. ACM, 92–104.

[16] Facebook Research. 2018. Deploying at Production Scale. https://www.facebook.com/pytorch/videos/vl.486038211901583/2142030806118038/?type=1. (2018).

[17] Jeremy Fowers, Kalin Ovtcharov, Michael Papamichael, Todd Massengill, Ming Liu, Daniel Lo, Shlomi Alkalay, Michael Haselman, Logan Adams, Mahdi Ghandi, Stephen Heil, Prerak Patel, Adam Sapek, Gabriel Weisz, Lisa Woods, Sitaram Lanka, Steven K. Reinhardt, Adrian M. Caulfield, Eric S. Chung, and Doug Burger. 2018. A Configurable Cloud-scale DNN Processor for Real-time AI. In *Proceedings of the 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE Press, Piscataway, NJ, USA, 1–14. https://doi.org/10.1109/ISCA.2018.00012

[18] Yijin Guan, Zhihang Yuan, Guangyu Sun, and Jason Cong. 2017. FPGA-based accelerator for long short-term memory recurrent neural networks. In *Design Automation Conference (ASP-DAC), 2017 22nd Asia and South Pacific*. IEEE, 629–634.

[19] Song Han, Junlong Kang, Huizi Mao, Yiming Hu, Xin Li, Yubin Li, Dongliang Xie, Hong Luo, Song Yao, Yu Wang, et al. 2017. Ese: Efficient speech recognition engine with sparse lstm on fpga. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 75–84.

[20] Awni Y. Hannun, Carl Case, Jared Casper, Bryan Catanzaro, Greg Diamos, Erich Elsen, Ryan Prenger, Sanjeev Satheesh, Shubho Sengupta, Adam Coates, and Andrew Y. Ng. 2014. Deep Speech: Scaling up end-to-end speech recognition. *CoRR* abs/1412.5567 (2014). http://arxiv.org/abs/1412.5567

[21] K. Hazelwood, S. Bird, D. Brooks, S. Chintala, U. Diril, D. Dzhulgakov, M. Fawzy, B. Jia, Y. Jia, A. Kalro, J. Law, K. Lee, J. Lu, P. Noordhuis, M. Smelyanskiy, L. Xiong, and X. Wang. 2018. Applied Machine Learning at Facebook: A Datacenter Infrastructure Perspective. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 620–629. https://doi.org/10.1109/HPCA.2018.00059

[22] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Computation* 9, 8 (Nov. 1997), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

[23] Akshay Jain, Mahmoud Khairy, and Timothy G Rogers. 2018. A Quantitative Evaluation of Contemporary GPU Simulation Methodology. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 2, 2 (2018), 35.

[24] Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre-luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, C. Richard Ho, Doug Hogberg, John Hu, Robert

Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Daniel Killebrew, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, Jonathan Ross, Matt Ross, Amir Salek, Emad Samadiani, Chris Severn, Gregory Sizikov, Matthew Snelham, Jed Souter, Dan Steinberg, Andy Swing, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox, and Doe Hyun Yoon. 2017. In-Datacenter Performance Analysis of a Tensor Processing Unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA)*. ACM, New York, NY, USA, 1–12. https://doi.org/10.1145/3079856.3080246

[25] Mahmoud Khairy, Akshay Jain, Tor M. Aamodt, and Timothy G. Rogers. 2018. Exploring Modern GPU Memory System Design Challenges through Accurate Modeling. *CoRR* abs/1810.07269 (2018). arXiv:1810.07269 http://arxiv.org/abs/1810.07269

[26] Farzad Khorasani, Hodjat Asghari Esfeden, Nael Abu-Ghazaleh, and Vivek Sarkar. 2018. In-Register Parameter Caching for Dynamic Neural Nets with Virtual Persistent Processor Specialization. In *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 377–389.

[27] Rakesh Komuravelli, Matthew D Sinclair, Johnathan Alsop, Muhammad Huzaifa, Maria Kotsifakou, Prakalp Srivastava, Sarita V Adve, and Vikram S Adve. 2015. Stash: Have your scratchpad and cache it too. In *ACM SIGARCH Computer Architecture News*, Vol. 43. ACM, 707–719.

[28] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*. 1097–1105.

[29] Andrew Lavin. 2015. maxDNN: An Efficient Convolution Kernel for Deep Learning with Maxwell GPUs. *CoRR* abs/1501.06633 (2015). arXiv:1501.06633 http://arxiv.org/abs/1501.06633

[30] Quoc V Le, Jiquan Ngiam, Adam Coates, Abhik Lahiri, Bobby Prochnow, and Andrew Y Ng. 2011. On optimization methods for deep learning. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*. Omnipress, 265–272.

[31] Minjae Lee, Kyuyeon Hwang, Jinhwan Park, Sungwook Choi, Sungho Shin, and Wonyong Sung. 2016. FPGA-based low-power speech recognition with recurrent neural networks. In *Signal Processing Systems (SiPS), 2016 IEEE International Workshop on*. IEEE, 230–235.

[32] Tao Lei, Yu Zhang, Sida I Wang, Hui Dai, and Yoav Artzi. 2018. Simple Recurrent Units for Highly Parallelizable Recurrence. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. 4470–4481.

[33] Jonathan Lew, Deval Shah, Suchita Pati, Shaylin Cattell, Mengchi Zhang, Amruth Sandhupatla, Christopher Ng, Negar Goli, Matthew D. Sinclair, Timothy G. Rogers, and Tor M. Aamodt. 2018. Analyzing Machine Learning Workloads Using a Detailed GPU Simulator. *CoRR* abs/1811.08933 (2018). arXiv:1811.08933 http://arxiv.org/abs/1811.08933

[34] Sicheng Li, Chunpeng Wu, Hai Li, Boxun Li, Yu Wang, and Qinru Qiu. 2015. Fpga acceleration of recurrent neural network based language model. In *2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 111–118.

[35] Sharan Narang. 2016. DeepBench. https://svail.github.io/DeepBench/. (2016).

[36] Sharan Narang and Greg Diamos. 2017. An Update to DeepBench with a Focus on Deep Learning Inference. https://svail.github.io/DeepBench-update/. (2017).

[37] Sharan Narang, Erich Elsen, Gregory Diamos, and Shubho Sengupta. 2017. Exploring sparsity in recurrent neural networks. *arXiv preprint arXiv:1704.05119* (2017).

[38] Wajahat Qadeer, Rehan Hameed, Ofer Shacham, Preethi Venkatesan, Christos Kozyrakis, and Mark A Horowitz. 2013. Convolution engine: balancing efficiency & flexibility in specialized computing. In *ACM SIGARCH Computer Architecture News*, Vol. 41. ACM, 24–35.

[39] Md Aamir Raihan, Negar Goli, and Tor M. Aamodt. 2018. Modeling Deep Learning Accelerator Enabled GPUs. *CoRR* abs/1811.08309 (2018). arXiv:1811.08309 http://arxiv.org/abs/1811.08309

[40] Kanishka Rao, Hasim Sak, and Rohit Prabhavalkar. 2018. Exploring Architectures, Data and Units For Streaming End-to-End Speech Recognition with RNN-Transducer. *CoRR* abs/1801.00841 (2018). arXiv:1801.00841 http://arxiv.org/abs/1801.00841

[41] Matthew D. Sinclair, Johnathan Alsop, and Sarita V. Adve. 2015. Efficient GPU Synchronization without Scopes: Saying No to Complex Consistency Models. In *Proceedings of the 48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 647–659.

[42] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. 2016. Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. *CoRR* abs/1609.08144 (2016). arXiv:1609.08144 http://arxiv.org/abs/1609.08144

[43] Minjia Zhang, Samyam Rajbhandari, Wenhan Wang, and Yuxiong He. 2018. DeepCPU: Serving RNN-based Deep Learning Models 10x Faster. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*. USENIX Association, Boston, MA, 951–965. https://www.usenix.org/conference/atc18/presentation/zhang-minjia

[44] Feiwen Zhu, Jeff Pool, Michael Andersch, Jeremy Appleyard, and Fung Xie. 2018. Sparse Persistent RNNs: Squeezing Large Recurrent Networks On-Chip. *CoRR* abs/1804.10223 (2018). arXiv:1804.10223 http://arxiv.org/abs/1804.10223

[45] Feiwen Zhu, Jeff Pool, Michael Andersch, Jeremy Appleyard, and Fung Xie. 2018. Sparse Persistent RNNs: Squeezing Large Recurrent Networks On-Chip. *arXiv preprint arXiv:1804.10223* (2018).