Training Larger Models on TensorFlow without Additional GPU

Jinliang Wei[1], Aurick Qiao[1,4], Anand Jayarajan[3,2], Garth Gibson[2,1,3]
Vijay Vasudevan[5], Eric Xing[1,4]

[1]Carnegie Mellon University, [2]Vector Institute, [3]University of Toronto, [4]Petuum Inc., [5]Google

ABSTRACT

Across different applications, we have seen examples that models with larger capacity achieve better performance. However, the model capacity is ultimately restricted by the limited and expensive GPU memory. In this paper, we present a number of techniques that leverage cheap host memory to reduce GPU memory consumption during training in TensorFlow. Our techniques require no modifications to TensorFlow application programs and enable training models with 3.8× more (up to 2.5 Billion parameters) parameters on a single GPU.

1 INTRODUCTION

Over recent years, deep learning has achieved wide success in many application domains, such as image classification [4], object detection [2], machine translation [9] and speech recognition [5]. Over many different applications, it is observed that within the same model class, models with larger capacity (more parameters) achieve better performance. We present some examples found in recent deep learning literature in Table 1.

Due to the computational overhead of deep neural network training, researchers and practitioners often resort to hardware accelerators, such as GPUs for fast training. However, the model size is restricted by the limited and highly expensive GPU memory. GPUs that are most commonly used for deep learning training today are limited to 12 or 16 GB of memory. Fig. 1 compares DRAM price with the price of a number of desktop and server GPUs that are popularly used for neural network training, in terms of $ per MBytes of on-board memory. We observe that GPU price is not affected by the decreasing DRAM price and remains highly expensive.

Motivated by these challenges, in this paper, we present a mechanism to reduce GPU memory consmpution to enable training bigger models by leveraging the cheaper host memory. We implemented those techniques in TensorFlow, without requiring additional input or modifications to the application program [1].

[1]Currently our techniques don't support computation graphs that use dynamic control flow operators, though this extension should not require significant innovation

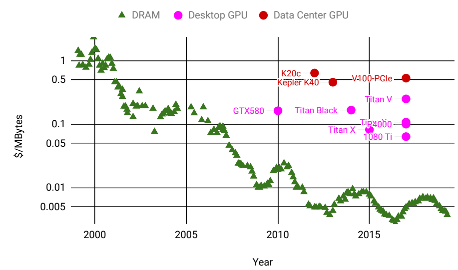

Figure 1: Comparing DRAM and GPU price (as dollars per MB of on-board memory).

We refer to our memory-optimized TensorFlow as *TensorFlowMem*. TensorFlowMem implements three main optimizations: 1) partitioned graph execution that executes teh computation graph in small partitions; 2) memory swapping that offloads tenosr values from GPU memory to host memory and prefetches them before they are needed; and 3) variable and constant placement that places stateful `Variable` and `Constant` operation on CPU memory and fetches them to GPU only when they are needed. We show that TensorFlowMem enables training a ResNet model of 1916 layers and a Mixture of Experts (MoE) model that has 2.9 billion parameters on a single GPU with 12 GB of memory. TensorFlowMem allows enables training RNNs on sequences of 2× longer sequences.

2 IMPLEMENTATION AND EVALUATION

2.1 Techniques and Implementation

Partitioned graph execution. TensorFlowMem implements a graph partitioning optimization pass. The graph partitioning pass performs a depth-first traversal of each device's computation graph and assigns nodes to fixed-size partitions according to the traversal order. Depth-first traversal makes best effort to consume intermediate results as soon as they are produced, instead of holding them in memory for longer durations.

Memory swapping. A graph partition may generate intermediate results that are consumed by partitions that are many sequential steps away in the computation schedule. TensorFlowMem temporarily offloads the intermediate result tensors

Model Class	Application	Metric	Scale By	Model Name	Scale	#Parameters	Performance
VGG [7]	CV	top-1 error	layers	VGG-16	16 layers	138 million	25.6
				VGG-19	19 layers	144 million	**25.5**
ResNet [4]	CV	top-1 error	layers	ResNet-50	50 layers	N/A	20.74
				ResNet-101	101 layers	N/A	19.87
				ResNet-152	152 layers	N/A	**19.38**
Mixture-of-Experts [6]	NLP	test perplexity	experts	MoE-32	32 experts	100 million	40.4
				MoE-4096	4096 experts	4.4 billion	30.9
				MoE-65536	65536 experts	9.2 billion	**28.9**
ResNet-40 [10]	CV	test error	widenning	N/A	$1\times$ wide	0.6 million	6.85
				N/A	$4\times$ wide	8.9 million	4.97
				N/A	$8\times$ wide	35.7 million	**4.17**

Table 1: Scaling model capacity via different ways. Results are collected from existing literature.

to host memory and prefetches them by adding `SwapOut` and `SwapIn` nodes. Sequential execution across partitions makes it easy to determine when operations are executed and place `SwapOut` nodes in proper partitions and prefetch CPU tensors shortly before they are needed.

Operation placement. A computation graph may contain `Variable` and `Constant` nodes that are stateful operations.These nodes are typically packaged with computation operations that use them as an integral building block by higher level programming interfaces, such as Keras [3]. When the application program places the computation operation on GPUs, `Variable`s and `Constant`s are implicitly placed on GPUs as well due to limited programming flexibility. Constant folding folds a subgraph into a `Constant` operation. The generated `Constant` operation are placed on the same computing device as the computation operations. TensorFlowMem places `Variable` and `Constant` nodes on CPU and loads their value to GPU when needed.

System	MB	#Layers	#Param.	TPut.
TensorFlow	16	504	172 Million	11.43
TensorFlowMem	16	1916	697 Million	1.76
TensorFlowMem	32	1001	385 Million	4.04

Table 2: Maximum ResNet model size that can be trained on a single Titan X GPU and computation throughput (TPut.) with different mini-batch size (MB).

2.2 Evaluation Results

In this section, we present a preliminary experimental evaluation of TensorFlowMem. All the experiments are conducted in a private cluster, where each machine has a 16-core Intel Xeon E5-2698B v3 processor with hyper-threading, 64GB of DRAM and a Titan X GPU with 12GB device memory.

Deeper ResNet. Similar to previous work [1, 8], we increase the model capacity of ResNet by scaling its number of layers. Specifically, we follow SuperNeuron [8] and increase the number of the third block. Our result is present in Table 2. Using the same mini-batch size of 16 images, TensorFlowMem scales to 1916 layers while TenosrFlow scales to only 504 layers. Using a mini-batch size of 32, TensorFlowMem scales to 1001 layers. Moreover, TensorFlowMem fails to scale to deeper ResNet because of running of host memory.

System	#Experts	#Param.	Throughput
TensorFlow	12 / MoE	0.66 B	7.8 pairs / sec
TensorFlowMem	48 / MoE	2.5 B	1.2 pairs / sec

Table 3: Maximum number of experts that can be trained on a single TitanX GPU. We use a batch size of 8 and graph partition size of 200 operations.

Mixture of Experts. We evaluate TensorFlowMem using Transformer w/MoE. Table 3 shows the maximum number of experts per MoE layer which can trained using TensorFlow, and TensorFlowMem. TensorFlowMem is able to train 48 experts for MoE layer, which is $4\times$ as many as vanilla TensorFlow[2]. We also find that the throughput decreases roughly linearly with respect to the number of experts when scaling up the MoE layers using TensorFlowMem.

Sequence Length	100	200	400	500	800
TensorFlow	1.15	2.3	4.64	OOM	OOM
TensorFlowMem	1.56	3.03	6.03	-	12

Table 4: Training iteration time vs. input sequence length

Longer Recurrence Sequences For RNNs, the sequence length is often limited by GPU memory size. TensorFlowMem

[2]We ran TensorFlow both with and without the Grappler memory swapping pass, but obtained the same result both times.

enables training RNNs on longer sequences with a small runtime overhead, which we demonstrate using Mozilla DeepSpeech, a statically unrolled RNN. Our experiments use a mini-batch size of 128 sentences and the partition size of TensorFlowMem is set to 5. Table 4 shows that TensorFlowMem can train DeepSpeech on sequences of length 800 while TensorFlow fails beyond sequence length of 400. Similar to ResNet, TensorFlowMem fails to scale to longer sequences due to the limited host memory. On the same sequence length, TensorFlowMem shows a runtime overhead of roughly 35%.

REFERENCES

[1] T. Chen, B. Xu, C. Zhang, and C. Guestrin. Training deep nets with sublinear memory cost. *ArXiv*, abs/1604.06174, 2016.

[2] J. Dai, Y. Li, K. He, and J. Sun. R-FCN: object detection via region-based fully convolutional networks. *CoRR*, abs/1605.06409, 2016.

[3] Francois Chollet. Keras. https://keras.io/, 2015.

[4] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[5] G. Hinton, L. Deng, D. Yu, G. Dahl, A. rahman Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. Sainath, and B. Kingsbury. Deep neural networks for acoustic modeling in speech recognition. *Signal Processing Magazine*, 2012.

[6] N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. Le, G. Hinton, and J. Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. 2017.

[7] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.

[8] L. Wang, J. Ye, Y. Zhao, W. Wu, A. Li, S. Leon Song, Z. Xu, and T. Kraska. Superneurons: dynamic gpu memory management for training deep neural networks. *ACM SIGPLAN Notices*, 53:41–53, 02 2018.

[9] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, J. Klingner, A. Shah, M. Johnson, X. Liu, Łukasz Kaiser, S. Gouws, Y. Kato, T. Kudo, H. Kazawa, K. Stevens, G. Kurian, N. Patil, W. Wang, C. Young, J. Smith, J. Riesa, A. Rudnick, O. Vinyals, G. Corrado, M. Hughes, and J. Dean. Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR*, abs/1609.08144, 2016.

[10] S. Zagoruyko and N. Komodakis. Wide residual networks. In E. R. H. Richard C. Wilson and W. A. P. Smith, editors, *Proceedings of the British Machine Vision Conference (BMVC)*, pages 87.1–87.12. BMVA Press, September 2016.