Experiences of Deep Learning Optimization on PAI

Siyu Wang
Alibaba Group, Beijing, China
siyu.wsy@alibaba-inc.com

Lanbo Li
Alibaba Group, Beijing, China
lanbo.llb@alibaba-inc.com

Jun Yang
Alibaba Group, Beijing, China
muzhuo.yj@alibaba-inc.com

1. Introduction

With the advent of big data, easy-to-get General Purpose Graphical Processing Unit (GPGPU) and recent progresses in modeling techniques, deep learning is becoming more and more important in data-oriented application such as as computer vision[12][9], speech recognition[10] and NLP[4][17]. Due to its inherent complexity, deep learning models usually bring challenges to offline training for both computation speed and memory usage[17][15][14].

There are already some off-the-shelf open-source deep learning framework such as MxNet[2] , TensorFlow[1] and etc. However, due to the fast evolving model architectures and specific in-house modeling requirement, dedicated effort is still necessary to optimize deep learning execution. In Alibaba, Platform of Artificial Intelligence(PAI), a large-scale machine learning platform, is developed to ease the use of machine learning by engineers and scientists. In PAI, lots of effort is allocated for optimizing the execution of deep learning models. For offline training, re-computation and CPU/GPU memory swap-out/swap-in strategies are employed to improve GPU memory usage. Practical placement tricks and model-oriented distributed strategy(such as heuristic-based model average) are also carefully designed to ensure the training process can be accelerated via distributed execution. All these tactics and strategies are put together to improve the productivity of deep learning jobs on PAI. Due to its popularity, all the optimizations are based on and integrated into TensorFlow. However, these strategies are also general enough and can be easily migrated to other deep learning frameworks.

2. Memory Optimization

In recent years, the evolution of model architecture has a "deeper and wider" trend. For example, "ResNet" [9] consists of more than 152 and even up to 1001 neuron layers and "DenseNet" [11] is composed of 162 layers. As a result of this, training deeper and wider models require significant memory consumption which easily exceeds single GPU capacity. Training deep learning model consists of forward and backward propagation phases. During backward propagation, gradients for model weights is computed which depend upon the intermediate results generated during forward pass. These intermediate results usually occupy a huge amount of GPU memory. [3] proposed to drop intermediate feature maps during forward pass and do re-computation during backward pass when necessary to reduce GPU memory consumption. This idea is also applied on Recurrent Neural Network(RNN) models with employing dynamic programming algorithm[8].

In this paper, based on TensorFlow's computation graph design philosophy, two optimization strategies are designed and integrated seamlessly, one for general models and the other for sequential models specifically. For general cases, "swap in/out" strategy for intermediate results [13] is employed. For sequential models, model-oriented "drop and re-computation" design [8] are taken into consideration. Intermediate results are dropped and re-computed for both ordinary sequential time-step and attention mechanism [6] with significant memory saving.

In our experiments, "swap in/out" is applied to ResNet-50 and Inception-v3 [16] on ImageNet [5] data-set. For RNN based attention model, we use "drop and re-computation" strategy. Our experiment settings are described in A.1 and results are showed in Table 1.

Table 1. Memory optimization results.D & R refers to "Drop and Re-computation" for short. B_{base} and B_{opt} refer to the maximum batch size allowed on single GPU before and after applying optimization respectively, the larger the better.

Strategy	Model	B_{base}	B_{opt}
Swap in/out	ResNet-50	144	664(+361%)
	Inception-v3	208	548(+163%)
D & R	DeepNMT	11	290(+1630%)
	Language Model	750	1500(+100%)

SOSP'17 October 29–31, 2017, Shanghai, China

3. Training Acceleration Optimization

In this section, some practical tricks will be shared about graph placement and model parameter partitioning to speed up training process. Additionally, model average combined with linear learning rate rule will also be proposed to address the distributed scalability issue. It should be noted that in our experiments up to 8 GPUs are used for a single training job mainly because this is quite enough for most training requirments in our real business.

3.1 Placement and Partitioning Tricks

In practice, we have found that two simple tricks would bring significant performance improvement.

- Placing data pre-processing operations on CPU.
- Partitioning model parameters into small shards across multiple server instances instead of just single one .

The first trick reduces unnecessary memory copies between host CPU and GPU devices, while the second one ensures communication across nodes in a more balanced way. Table 2 shows the benchmark results on AlexNet and ResNet-32. Experiment settings are described in A.2.

Table 2. Practical tricks benchmark results. T_r, T_p and T_c refer to one iteration time without any tricks, only with IO placement trick and with both IO placement and variable partition tricks respectively, the less the better.

Model		GPU cards number			
		1	2	4	8
AlexNet	T_r(s)	0.952	0.539	0.537	0.78
	T_p(s)	0.807	0.52	0.518	0.757
	T_c(s)	0.807	0.451	0.279	0.198
ResNet-32	T_r(s)	0.141	0.144	0.157	0.158
	T_p(s)	0.079	0.083	0.89	0.091
	T_c(s)	0.079	0.083	0.086	0.088

The placement optimization on ResNet-32 scale much better mainly because it reduces more memory copies between CPU and GPU than it is on AlexNet. Also the fewer parameters of RaesNet-32 leads to less communication time.

3.2 Model Average

In distributed training, model average strategy can reduce communication overhead significantly due to that it reduce the communication frequency aggressively. However, it will not bring significant convergence speedup without applying a linear learning rate rule[7]. This is because the contributed gradients from each sample batch will be averaged by node number after applying a model average operation. In order to make compensation for the weakened contributions, it is essential to increase learning rate proportionally to the node number.

To integrate model average into TensorFlow seamlessly, model average is formulated as a graph construction problem. Actually in our design model average consists of three execution stages. Firstly, each worker pulls the global parameters from parameter server to the local worker. Secondly, each worker trains their own model replica within fixed number of steps. Finally, each worker pushes their latest local parameters to the parameter server and average them. These three stages are combined into one graph in TensorFlow and each stage can be viewed as a sub-graph. Each sub-graph is executed separately correspondingly.

Figure 1 illustrates the model average design in TensorFlow.

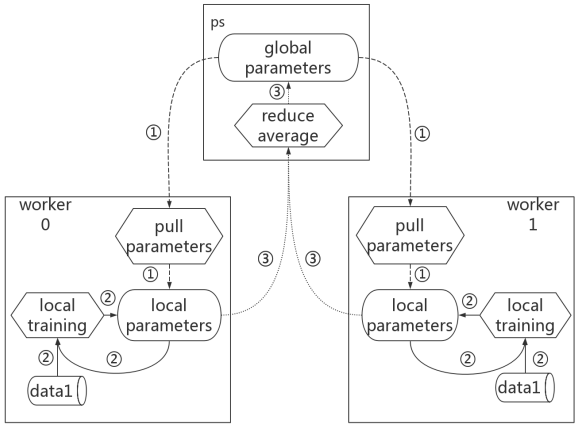

Figure 1. Model average design in TensorFlow

Experiments of model average are taken with a public ResNet-32 model and an in-house NMT model on PAI platform. The experiment setting is described in A.3. Table 3 shows the result.

Table 3. Model average benchmark results, C_t refers to how long it takes to converge, the less the better.

	Model	GPU cards number			
		1	2	4	8
C_t(min)	ResNet	137.8	69.35	37.6	21.5
speedup		1X	1.98X	3.67X	6.40X
C_t(min)	NMT	1352.8	745.4	375.9	218.5
speedup		1X	1.81X	3.60X	6.19X

4. Conclusion

Deep learning optimization is a fast evolving area from both research and industry perspective. More optimizations are still necessary for improving productivity. How to make distributed training more easily for ordinary research users who are in lack of distributed implementation experiences? How to design new model which is more friendly with computation devices and distributed execution scenario? How to distill offline trained models into compact version in principal way with better computation and energy performance for online inference? These are all interesting and practical problems. We hope more experiences and progress could be shared in the future.

References

[1] 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*, Vol. 16. 265–283.

[2] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. 2015. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *arXiv preprint arXiv:1512.01274* (2015).

[3] Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. 2016. Training deep nets with sublinear memory cost. *arXiv preprint arXiv:1604.06174* (2016).

[4] Ronan Collobert and Jason Weston. 2008. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*. ACM, 160–167.

[5] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE, 248–255.

[6] Bahdanau Dzmitry, Cho Kyunghyun, and Bengio Yoshua. 2014. Neural Machine Translation by Jointly Learning to Align and Translate. *arXiv preprint arXiv:1409.0473* (2014).

[7] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. 2017. Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour. *arXiv preprint arXiv:1706.02677* (2017).

[8] Audrunas Gruslys, Rémi Munos, Ivo Danihelka, Marc Lanctot, and Alex Graves. 2016. Memory-efficient backpropagation through time. In *Advances in Neural Information Processing Systems*. 4125–4133.

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 770–778.

[10] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. 2012. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine* 29, 6 (2012), 82–97.

[11] Gao Huang, Zhuang Liu, Kilian Q Weinberger, and Van Der Maaten Laurens. 2017. Densely Connected Convolutional Networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[12] 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.

[13] Minsoo Rhu, Natalia Gimelshein, Jason Clemons, Arslan Zulfiqar, and Stephen W. Keckler. 2016. Virtualizing Deep Neural Networks for Memory-Efficient Neural Network Design. *arXiv* (2016).

[14] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. 2017. Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer. *arXiv preprint arXiv:1701.06538* (2017).

[15] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 1–9.

[16] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. 2016. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2818–2826.

[17] 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* (2016).

A. Experiment Settings

This appendix describes the experiment settings in each section.

A.1 Memory optimization

- Hardware: NVIDIA Tesla P100
- Software: CUDA 8.0 + cuDNN V5
- Framework: TensorFlow
- ImageNet Models: Inception-v3 and ResNet-50
- deep NMT Models: in-house deep NMT model with 40k vocabulary size, 4 LSTM layers with hidden number 1000 and timestep 30, 2 attention layers
- language Models: lstm-based language model with 10k vocabulary size, 4 LSTM layers with hidden number 2048 and timestep 35

A.2 Placement&Partitioning Tricks

- Hardware: NVIDIA Tesla P100
- Software: CUDA 8.0 + cuDNN V5
- Network Protocol: RDMA
- Framework: TensorFlow
- Models: ImageNet for AlexNet and Cifar-10 for ResNet-32
- Batch Size: Strong scaling with total batch size 512 for AlexNet and weak scaling with batch size 128 on each worker for ResNet-32

A.3 Model Average

- Hardware: NVIDIA Tesla P100

- Software: CUDA 8.0 + cuDNN V5
- Network Protocol: RDMA
- Framework: TensorFlow
- Models: Cifar-10 for ResNet-32 and in-house NMT model
- Batch Size: 128 for ResNet-32 and 160 for in-house NMT model
- Model Average Frequency: 10 step for ResNet-32 and 1 epoch for in-house NMT model