Understanding the Resource Demand Differences of Deep Neural Network Training

Jiangsu Du(✉), Xin Zhu, Nan Hu, and Yunfei Du

School of Data and Computer Science, Sun Yat-Sen University, Guangzhou, China
dujs@mail2.sysu.edu.cn

Abstract. More deep neural networks (DNN) are deployed in the real world, while the heavy computing demand becomes an obstacle. In this paper, we analyze the resource demand differences of DNN training and help understand its performance characteristic. In detail, we study both shared-memory and message-passing behavior in distributed DNN training from layer-level and model-level perspectives. From layer-level perspective, we evaluate and compare basic layers' resource demand. From model-level perspective, we measure parallel training of representative models then explain the causes of performance differences based on their structures. Experimental results reveal that different models vary in resource demand and even a model can have very different resource demand with different input sizes. Further, we give out some observations and recommendations on performance improvement of on-chip training and parallel training.

Keywords: Deep neural network training · Performance · Resource demand differences

1 Introduction

Over the last few years, deep learning (DL) achieves great success in many domains. New deep learning (DL) applications are constantly developed and deployed to real-world utility [4]. New requirement that provides high performance under limited budgets is emerged.

In this paper, we uncover resource demand differences of DNN models from which people can understand the resource demand features of all kinds of models. In order to have a comprehensive understanding, we analyze models in a divide-conquer style, from layer-level and model-level perspectives. From layer-level perspective, we first abstract training process of DNN and measure the floating point operands (FLOPs), memory consumption, and communication amount of basic layers. Moreover, two metrics are designed to compare their resource demand differences. From model-level perspective, we evaluate overall throughput with different batch sizes and interconnection networks. Then an

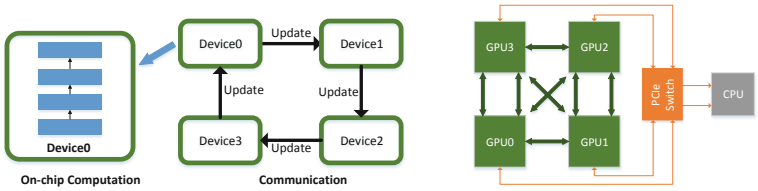
S. Wen et al. (Eds.): ICA3PP 2019, LNCS 11945, pp. 673–681, 2020.
https://doi.org/10.1007/978-3-030-38961-1_56

analysis is given based on their structure. We provide readers a comprehensive insight on DNN training, and make some important observations and recommendations on performance improvement of DNN on-chip computing and parallel computing.

2 Methodology

2.1 Training Simplification

Based on a careful technical survey and the node usage of a commercial V100 GPU cluster locating at national supercomputing center in Guangzhou, we choose data parallelism, synchronous stochastic gradient descent (SGD), and all-reduce methods as our experimental object since their effectiveness and popularity. Notably, our focus is the feature of models, and the simplification is to make our analysis intuitive. As shown in Fig. 1(a), under the configuration above, DNN training can be divided into on-chip computation and off-chip communication. In each iteration, each device has a complete model copy and runs both feed-forward and back-propagation locally. After all devices complete computing, updates will be aggregated for next iteration.



(a) Training Simplification (b) Topology of DGX Station

Fig. 1. Parallel Training and DGX Topology (Color figure online)

2.2 Layer-Level Perspective

The training of DNN includes two processes, feed-forward and back-propagation. Generally, a DNN is made up by several different layers and the computing process between upper and lower layers is independent. Thus, the next layer cannot start operating until finishing the previous layer. Therefore, the overall training process can be divided by layers and studied independently.

Basic layers mainly used today are Fully Connected Layer (FCL), Convolutional Layer (CONVL) and Recurrent Layer (RCL). Their static structure can be easily learned from online resources today. For RCL, the basic RCL and two variants, LSTM and GRU are considered.

Here we analyze runtime resource demand which is floating point unit, memory and interconnection. So we measure **FLOPs, memory consumption, and communication amount** to reflect the demand. At first, we identify what resource demand should be included for a layer. For FLOPs demand, it is easy to distinguish. For memory demand, the boundary is not that clear. The memory of a layer consists of input placeholder, newly requested memory by operations, and weights. Input placeholder is also the output of the upper or lower layer, so it is not counted in. In this way, we only include newly requested memory and weights as a layer's memory demand. Also it is common to reuse memory requested in feed-forward for back-propagation, so here we don't count twice. Notably, the intermediate result produced by feed-forward is called feature map and that of back-propagation is called gradient map. As for communication demand, because almost all communication overhead comes from weight synchronization, so we represent communication demand by weight amount.

We implement each basic layer in Tensorflow and evaluate using TFprofiler. Because minibatch SGD can largely increase the concurrency in today's multi-core or many-core architecture, we evaluate the demand with different batch sizes. Additionally, different configurations of layers are taken into account.

Moreover, two metrics are defined to compare resource demand. The first metric is based on two facts. The first fact is that the memory access intensity of dominant operations in these layers are similar. Second, because of the compute dependency, the FLOPs can not directly determine the running time. However, for RCL, the influence of dependency goes weaker as batch size becomes large. The first metric is floating point operands per weight (FOPP). It can reflect how sensitive is a model to interconnect performance. The mathematical expression is as follows:

$$FOPP = \frac{floating\,point\,operands}{weights \times batch\,size} \tag{1}$$

The second metric is instant floating point operands per memory (IFOPM), it reflects the demand ratio of floating point unit and memory size. As for RCL, it should be additionally divided by time step since the FLOPs of different time step cannot be computed simultaneously. The mathematical expression is as follows:

$$IFOPM = \frac{floating\,point\,operands}{memory\,usage\,(\times time\,step)} \tag{2}$$

2.3 Model-Level Perspective

We analyze the resource demand differences of models selected through observing their performance change with different memory usage and interconnects. Memory usage is achieved by setting different batch sizes. As for different interconnect, we switch between NvLink and PCIe. Figure 1(b) displays the topology of DGX Station. It has 1 CPU and 4 GPUs. Each GPU can access other GPUs by NvLink (green lines) or PCIe Gen3 $\times 16$ (orange lines). According to our measurement, the bandwidth of Nvlink is about $5\times$ of PCIe and latency is only

1 eighth. Obviously, there is a great difference between these two interconnection networks.

3 Evaluation and Analysis

3.1 Environmental Setup

The software we use: Ubuntu 16.04.4 LTS, CUDA 10, NCCL 2.4.2, cuDNN 7.4.2, Tensorflow v1.11, Pytorch 1.0. DGX Station is equipped with a Intel Xeon E5-2698 V4 CPU and 4 Tesla V100 (32 GB) with NVLink.

3.2 Basic Layer Result

Fully Connected Layer. We configure FCL with different batch size and neuron number, and evaluate corresponding weight amount, FLOPs and memory usage. From the result, we can observe that FLOPs increase proportionally with both layer size and batch size. The dominant operation in FCL is matrix multiplication which accounts for more than 99%. For memory, it increases proportionally with layer size and a little with batch size. Memory demand for FCL is from memory newly requested by matrix multiplication and weights. For matrix multiplication, it needs one copy of weight and only request new memory for feature map in feed-forward. When using larger batch size, only feature map will increase. However, the variable number of feature map is only equal to layer size, so the memory consumed by weights is thousands of times larger than that of feature map.

Insights: Weights occupy most memory demand in FCL training and it only brings a little memory increase with larger batch size.

Convolutional Layer. We configure CONVL with different batch size and kernel size, and evaluate corresponding weights, FLOPs and memory usage. Our result presents that FLOPs and memory demand are almost proportional to batch size. For FLOPs, the dominant operation is Conv2D which accounts for more than 99.5%. For memory demand, feature map occupied most of newly requested memory. Not like FCL, memory usage of feature map is much larger than that of weights in CONVL.

Insights: Intermediate result occupies most memory demand in CONVL training. If memory size becomes a limitation for DNN training in GPU. CONVL can be the primary structure to be considered when reducing memory demand by re-calculating feature maps.

Recurrent Layer. We configure RCL with different batch size, neuron number and time step, and evaluate corresponding weights, FLOPs and memory usage. It can be observed that different RCLs demonstrate very similar trends

on FLOPS and memory. For weight number, they are only influenced by hidden layer size. For FLOPs, matrix-related operations dominate the overall computational complexity and they occupy more than 99%. Notably, the weight amount and FLOPs of these three RCLs are about 1:3:4. For memory, it is much more complicated than FCL and CONVL. Memory is not mainly requested by a single operation. In RCLs, newly requested memory comes from element-wise, matrix-vector multiplication, and data movement operations. The increase of memory is only proportional to time step and slower than a linear relation with hidden layer size and batch size. Based on the profiling result, these implementations will take three copies of weights. Even so, weights only contribute to a small percentage of memory usage and the intermediate result is dominant.

Comparison. The comparison uses metrics, FOPP and IFOPM, raised above. To make the result easy-observable, values are normalized.

For FOPP, FCL and CONVL fluctuate at a stable value. FCL is about 0.006 and CONVL is about 6. In terms of RCL, the metric of basic RCL, GRU, and LSTM only change with time step. If we divide FOPP of RCL by time step, they are similar with FCL at 0.006. As we investigate in complete applications, time step is the length of human sentence in general, so FOPP of these layers: $FCL >> BasicRNN \approx LSTM \approx GRU >> CONVL$.

Insights: FCL or RCL, especially FCL, usually contribute to more weights and less computation comparing to CONVL.

For IFOPM, all these layers change in a wide range. We explore their range based on evaluation and theoretical analysis. Firstly, variables of a FCL are input size, output size, and batch size. As claimed above, both memory demand and FLOPs are proportional to input size. FLOPs are proportional to output size and memory demand is almost not related to output size. For batch size, it ranges widely from 16 to 1024 or even larger. So, IFOPM of FCL is approximately from 12 to 756 (even larger and mainly around 100). Secondly, variables of a CONVL are kernel size, kernel number, batch size, input size. We can know that input size, batch size, and kernel number only slightly influence this metric. For kernel size, it is quadratic to FLOPs and only influence memory demand a little. The biggest kernel size yet we know is 11 and it cannot be smaller than 2. Also, a kernel is sometimes 3 dimensions and IFOPM should be multiplied with the channel number. So, IFOPM of CONVL is approximately from 2 to 183 (even larger). Thirdly, for RCL, IFOPM of three variants are similar and always $LSTM > GRU > BasicRCL$ when using same configuration. We consider hidden layer size is from 16 to 1024, batch size is from 16 to 512, and time step is from 5 to 40. Then, basic RCL is from 0.17 to 23, GRU is from 0.17 to 24.5, and LSTM is from 0.26 to 25.7. Comparing these three layer types, the rank of IFOPM is $FCL \geq CONVL > LSTM > GRU > BasicRCL$ in most circumstances.

Insights: A layer with different input size or configuration can vary in resource demand. FCL has much more weights than CONVL, but the IFOPM of FCL

can be similar or even larger than CONVL, which is different to our initial impression. Additionally, for a device, RCL occupies larger memory then it can use up floating point unit.

3.3 Model Result

This part we evaluate representative models that achieve competitive accuracy in their domains. Models are listed in Table 1.

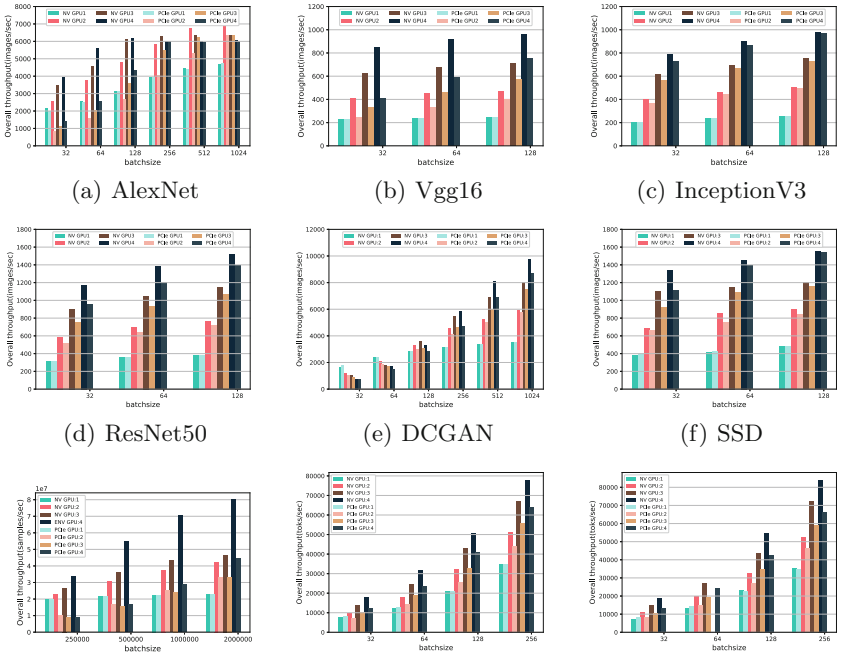
Table 1. Domains, models, datasets, and frameworks

Domains	Models	Dominant layer	Framework	Dataset
Image classification	AlexNet	CONVL, FCL	Tensorflow	ImageNet-1k
	Vgg16	CONVL, FCL	Tensorflow	ImageNet-1k
	ResNet50	CONVL	Tensorflow	ImageNet-1k
	InceptionV3	CONVL	Tensorflow	ImageNet-1k
Object detection	SSD [2]	CONVL	Pytorch	COCO
Recommendation	NCF [1]	FCL	Pytorch	MovieLens
Adversarial learning	DCGAN [3]	CONVL	Pytorch	LSUN
Machine translation	Seq2Seq [5] (GRU)	GRU	Pytorch	WMT16
	Seq2Seq (LSTM)	LSTM	Pytorch	WMT16

We display our results in Fig. 2 and further extract features in Fig. 3(a) and Fig. 3(b). Figure 3(a) is the performance improvement rate when expanding batch size. In other words, it is the ratio of performance with two neighborhood batch size when using a single GPU. Figure 3(b) shows the performance ratio of 4 GPUs with different interconnection networks.

AlexNet, Vgg16, InceptionV3, ResNet50. We first compare the results of image classification models. For AlexNet, it has 8 layers (5 convolutional layers and 3 fully connected layers). After simple calculation, almost all the weights come from fully connected layer. For Vgg16, it has 16 layers (3 FCLs) and most of weights still come from fully connected layer. For ResNet50, only one fully connected layer is used in ResNet. In this way, weights are not mainly contributed by fully connected layer. For InceptionV3, it completely remove fully connected layer.

Moving on to the evaluation, Fig. 2(a), (b), (c), and (d) show the result of these four models. Initially, we focus on the on-chip performance with different batch size. When batch size is small, the expansion of batch size can bring considerable performance improvement, and it becomes weak when batch size goes large. As for AlexNet, because of simplicity, it obtains good performance improvement at beginning then declines quickly. The growing rate of other three models is very limited, especially for Vgg16. From the evaluation, we can predict there exist a saturation point of floating point unit and the performance will not keep increasing with batch size. In other words, it uses up floating point unit.



(a) AlexNet (b) Vgg16 (c) InceptionV3

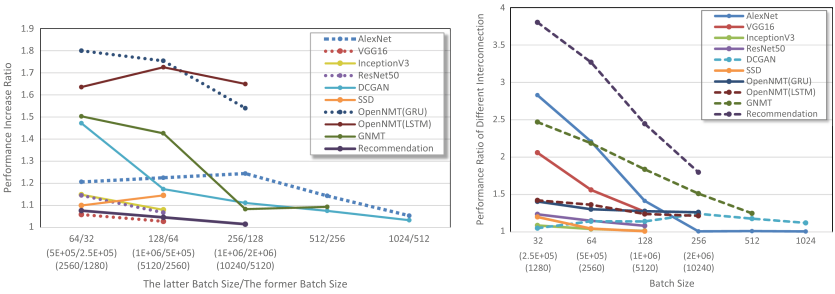
(d) ResNet50 (e) DCGAN (f) SSD

(g) NCF (h) OpenNMT LSTM (i) OpenNMT GRU

Fig. 2. Model performance display

Moving on to multi-GPU training, we can observe that all these models gain performance improvement when using large batch size, because the communication frequency is relatively reduced. However, the batch size cannot be increased infinitely since it will damage convergence speed. From Fig. 3(b), models show different sensitivity to interconnection. For AlexNet, it is influenced largely switching to weak interconnection. For Vgg16, the influence of weak interconnection is also huge (about 21%) but much better than AlexNet. For InceptionV3 and ResNet50, the bad interconnection performance damages only a little performance (about 1.4% and 7.6%). These ratio is calculated when using 4 GPUs and the largest batch size.

DCGAN. DCGAN uses two CNNs as the core of model. Although the training process is more complicated than pure CNN models, its training can be simply considered as the addition of two CNNs. The implementation uses four convolutional layers as generator network and five convolutional layers as discriminator network. It removes all fully connected layer and pooling layer. Figure 2(e) displays the evaluation result. For single GPU training, its performance improvement rate is quite high at the beginning comparing with other CNN based models since it is a very small model which can hardly consume much resource. Then the improvement rate gradually reduces to a normal level. For multi-GPU training,



(a) On-chip Computation (b) Parallel Computation

Fig. 3. Performance change summary

it is not very sensitive to interconnect performance. It experiences about 11% performance decline when using 4 GPUs at batch size 1024.

Single Shot MultiBox Detector. SSD can use ResNet, Vgg, and other classical CNN as its backbone. In our implementation, ResNet34 is used. Besides ResNet34, there exists some other structure which contributes extra time.

Figure 2(f) displays the evaluation of SSD. Obviously, in Fig. 3(a) and (b), it shows very similar trend with InceptionV3 and ResNet50.

Neural Collaborative Filtering. NCF can be divided into 4 layer types: input layer, embedding layer, neural CF layer, and output layer. In our implementation, all these layer are substantially FCLs. For single GPU training, as is shown by Fig. 3(a), it only improves a little, which can validate that FCL has very high IFOPM. As for multi-GPU training, even using large batch size, it experiences a 44.4% decline when switching interconnection. It is extremely sensitive to interconnect since it is almost fully made up by FCLs.

Insights: All previous models use CONVL and FCL as their main structure. FCL usually contributes most weights and CONVL contributes most FLOPs in a CNN. For on-chip performance, they quickly occupy all floating point unit when increasing batch size. For parallel performance, FCL hugely influences the scalability of training and CONVL-dominant models show slight performance decline when switching to weak interconnection. For FCL-dominant models, data parallelism can gain even no improvement if only PCIe provided.

Seq2Seq (GRU), Seq2Seq (LSTM). These two models are dominated by RCLs. Here we use the Seq2Seq demo provided officially by OpenNMT. The demo uses 2 RCLs as encoder and another 2 RCLs as decoder (500 hidden size). Also, users can choose RCL type, LSTM or GRU.

Figure 2(i), and (h) demonstrate the results. From Fig. 3(a), they occupy the top 2 places. Although they experience a decline when batch size increases, the improvement rate is still very high. In other words, they gain more improvement when increasing batch size. In other words, it is difficult for RCLs to occupy all floating point units with small batch size. For multi-GPU training, even

with large batch size, weak interconnection still damage overall performance a lot (20.8% for OpenNMT (GRU), 17.8% for OpenNMT (LSTM)). 3-GPU training with Nvlink sometimes is even better than 4-GPU training with PCIe. Additionally, comparing Seq2Seq(GRU) and Seq2Seq(LSTM), it validates that LSTM consumes more resources than GRU.

Insights: For on-chip performance, RCL based model needs a large batch size to occupy all floating point unit which leads to higher memory requirements. For parallel performance, RNN heavily depends on interconnection performance. Additionally, for all models, a frequently mentioned but very important insight is that increasing batch size can largely improve the on-chip running time and decrease communication frequency.

4 Conclusion

DNN training has stepped into a new stage, which raised new challenges on improving performance and reducing cost. We try to uncover resource demand differences of DNN training. The work focuses on both on-chip computation and off-chip communication. To have an insight on the demand, we analyze from layer-level and model-level perspectives. The results reveal that there exist huge resource demand differences among models. In detail, FCL and RCL should contribute to much more communication overhead. FCL has much more weights than CONVL but it has similar or even larger FOPP. For RCL, because of computing dependency, it will create more intermediate results and RCL needs larger memory size to use up device's floating point unit. Based on these results, we make several important observations, which can provide guidance for designing software and hardware or simply purchasing new hardware.

Acknowledgement. This research was supported by the Natural Science Foundation of China under Grant NO. U1811464 and the Program for Guangdong Introducing Innovative and Enterpreneurial Teams under Grant NO. 2016ZT06D211.

References

1. He, X., Liao, L., Zhang, H., Nie, L., Hu, X., Chua, T.S.: Neural collaborative filtering. In: Proceedings of the 26th International Conference on World Wide Web, pp. 173–182. International World Wide Web Conferences Steering Committee (2017)
2. Liu, W., et al.: SSD: single shot multibox detector. In: Leibe, B., Matas, J., Sebe, N., Welling, M. (eds.) ECCV 2016. LNCS, vol. 9905, pp. 21–37. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46448-0_2
3. Radford, A., Metz, L., Chintala, S.: Unsupervised representation learning with deep convolutional generative adversarial networks. arXiv preprint arXiv:1511.06434 (2015)
4. Ratner, A., et al.: SysML: the new frontier of machine learning systems. arXiv preprint arXiv:1904.03257 (2019)
5. Sutskever, I., Vinyals, O., Le, Q.V.: Sequence to sequence learning with neural networks. In: Advances in Neural Information Processing Systems, pp. 3104–3112 (2014)