The 22nd Meeting on Image Recognition and Understanding

Reduce GPU Memory Usage of Training Neural Network by CPU Offloading

Weihao Zhuang[1,a)] Tristan Hascoet[1,b)] Ryoichi Takashima[1,c)]
Tetsuya Takiguchi[1,d)] Yasuo Ariki[1,e)]

Abstract

Recently, Convolutional Neural Networks (CNN) have demonstrated state-of-the-art results on various computer vision problems. However, training CNNs require specialized Graphical Processing Units (GPU) with large memory. Given the ubiquity of CNN in computer vision, optimizing the memory consumption of CNN training would have wide spread practical benefits. In this paper, we propose an algorithm to reduce the GPU memory needed to train neural network by offloading the input activation of hidden layers to the CPU: during the forward pass, we transfer input activations to the CPU upon computation to free up GPU memory, and transfer these activations back to the GPU when needed by the backward pass. Our algorithm neither change the architecture of neural networks, nor use any compression algorithm. On the VGG architecture, our algorithm achieves a reduction of 29.6% of the GPU memory usage while increasing the computation time by only 4%. On MobileNet, we reduce the memory consumption of training iterations by 68.4% with a minimal overhead of 14% in computation time.

1. Introduction

Deep convolutional neural networks have shown unprecedented progress in computer vison since Krizhevsky et al. [1] won the 2012 ILSVRC. Since then, CNN have been successfully applied to many different real-world applications. The backpropagation algorithm need the hidden layer activations to accumulate in live memory during the forward pass in order to compute the gradients of the weights during the backward pass, which creates a memory bottleneck. Hence, training state-of-the-art CNN has required special hardware with large memory capacity as typical desktop memory is too small for backpropagation training. Meanwhile, a number or resent work have demonstrated the benefits of large batch training [2], further increasing these memory requirements. Hence optimizing the memory usage of CNN training would enable both research on optimization and training on low-end GPU devices.

There is an inherent trade-off between memory consumption and computation time: gradient checkpointing methods [3] only store a fraction of the hidden activations and reconstructing the missing activations from the stored ones during the backward pass. Reversible Network (RevNet)[4] constrain the architecture of Residual Networks to invertible transformations so that each layer's input activations are reconstructed from their output during the backward pass.

In this paper we design a general algorithm to reduce the GPU memory usage of CNN training by offloading the input activation of hidden layers into CPU memory upon computation during the forward pass, and transferring these activations back to GPU as needed during the backward pass. The challenge is to minimize the additional time of CPU offloading. We present several tricks to efficiently parallelize computation and data transfer to reduce the memory cost with a minimal additional time. Our algorithm allows us to reduce by 34.4% and 48.7% the memory required to train the VGG and MobileNet architectures with a minimal time overhead of 1% and 4% respectively.

2. Related Work

Several approaches have been proposed to reduce memory usage and computation consumption. MobileNet

[1] Kobe University
[a)] 191x124x@stu.kobe-u.ac.jp
[b)] tristan.hascoet@gmail.com
[c)] rtakashima@port.kobe-u.ac.jp
[d)] takigu@kobe-u.ac.jp
[e)] ariki@kobe-u.ac.jp

[5] is a light neural network using depth-wise separable convolution to reduce computation complexity to perform inference on low performance device. Quantized Neural Network (QNN) [6] is designed to reduce memory by performing low-precision arithmetic operations.

Gradient checkpointing is an efficient trick for reducing CNN memory usage: it only stores a few of the hidden layers input activations during the forward pass and reconstruct the lost activations during the backward pass.

Gomez et al. propose a Reversible Residual Network (RevNet) architecture to reduce memory consumption. RevNet uses invertible residual modules to recompute the hidden activation of lower layers from those of higher layers. Hence, activations need not to be stored during the forward pass as they can be recomputed during the backward pass. RevNet offers an efficient memory and computation trade off than gradient checkpointing but imposes additional constraint on the network architecture.

In contrast, our algorithm does not add any additional computation nor architecture restrictions we propose to transfer the hidden activations of each layers from GPU to CPU memory. The key challenge for our algorithm is the efficient parallelization of data transfer and computations.

3. Methods

We introduce how to implement our algorithm in this section; Figure 1 illustrates a simple neural network architecture, in which orange circles represent successive layers of the neural network.

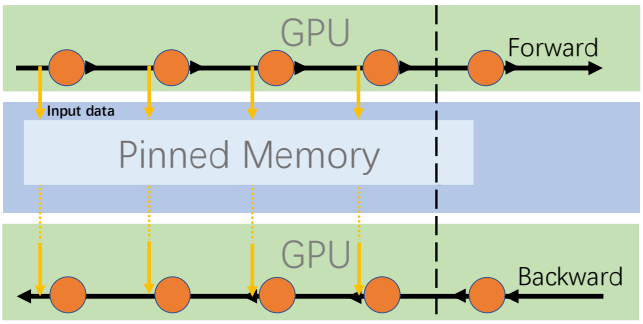


Fig. 1: Use pinned memory strove input activation of hidden layer

When we train CNN, the input activations will accumulate in memory during the forward pass. Each stored activation is stored only to be used for the computation of the layer weight gradients during the backward pass. Hence, the activations of lower layers can be temporarily freed from GPU memory during the computation of the forward and backward pass through the higher layers.

We propose to offload these activations to the CPU. For example, when the forward pass through the first layer completes, our algorithm offloads the input activation of first layer to CPU memory and forward its output to the second layer. In the backward pass, the input data of each layer is transferred back to GPU memory right before computation of the weight gradients.

There are two types of CPU memory: pageable memory and pinned memory are used to transfer data between CPU and GPU. In terms of transfer speed, using pinned memory is faster than pageable memory. Although pinned memory is providing high transfer speed, the time to allocate them are expensive. So, we allocate pinned memory beforehand, during the instantiation of the CNN.

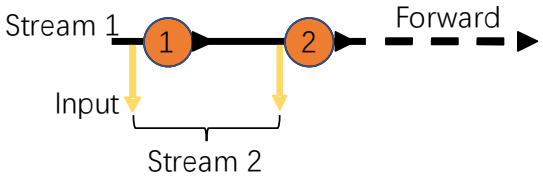


Fig. 2: Use another stream doing data transfer

The key problem we have to solve is how to overlap computation and data transfer so that, in the forward pass, we do not have to wait for the data transfer to CPU to finish before computing the next layer of the forward pass, which would slow down the training process. Similarly, during the backward pass, we would also like to avoid the additional time of waiting for the data transfer to complete before computing the gradient. Therefore, finding an efficient way to parallelize computation and data transfer is important.

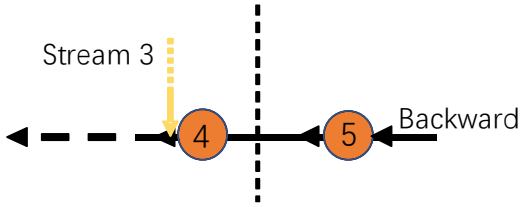


Fig. 3: Some tricks use in our algorithm in backward pass

In our implementation, we use multithreading to parallelize the CPU-side operations and CUDA streams for GPU-side data transfer.

As shown in Figure 2, we dedicate a CUDA stream to perform the computation and another CUDA stream to transfer the input activations to CPU during the forward

pass.

The parallelization of the backward pass is more complex than the forward pass because we cannot parallelize the computation and data transfer within one hidden layer as we need the input activation to reside in GPU memory before computing the weight gradients.

To improve parallelization, we synchronize the data transfer to GPU of the activation of a given layer with the computation of its upper layer. Figure 3 shows the example of the data transfer synchronization in layer 4 and 5 of our example network. This figure shows that the transfer of input activation of layer 4 to GPU is overlapped with the computation of the backward pass through layer 5.

4. Experiment

In this section, we discuss the performance and results of our implementation. Table 1 describes our hardware configuration:

Table. 1: Experiment configuration

CPU	Intel Core i5-8400 2.80GHz
GPU	GTX 1060 6GB
Motherboard	GIGABYTE B360M HD3
RAM	16GB DDR4 2400MHz

4.1 Transfer data in pinned memory and pageable memory

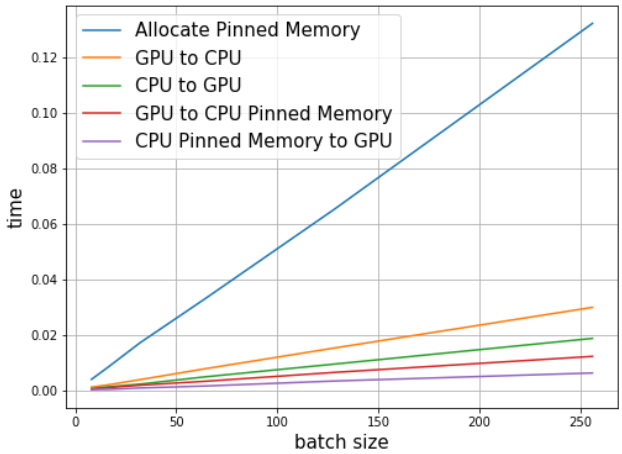


Fig. 4: Speed of transfer data between CPU and GPU with and without pinned memory

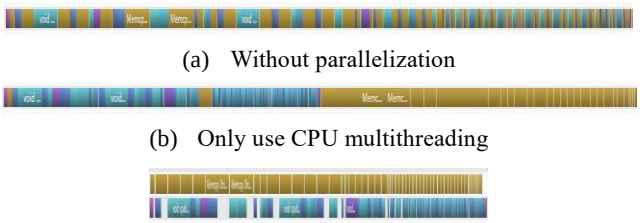
Figure 4 illustrated the transfer speed for different device communication. The timings presented represent the transfer time for a batch of half precision images 224 ×224×3with different batch sizes. Data transfer speed are measured for both pinned memory and pageable memory. Transfer times increase linearly with the batch size. Pinned memory is 2.5 times faster than pageable memory when transfer data from GPU to CPU and 3 times faster from CPU to GPU. However, allocating pinned memory is very slow so it is necessarily to allocate pinned memory before training.

4.2 Parallel computing and data transfer

Figure 5 illustrates the parallelization of data transfer and computation. Brown areas illustrate data transfer operations and blue areas illustrate computation operations. The top figure shows the sequence of operations performed without parallelization: computations and data transfer are executed sequentially so that the GPU cores spend a lot of idle time and the computation is slowed down by data transfer.

The figure 5b shows the inability of CPU multithreading to parallelize data transfer and computations

Figure 5c shows that CUDA streams successfully overlap computation and data transfer as we expect it. One stream is used for computation and another is used for data transfer, there are still some blank in computation because computation in forward pass faster than data transfer. This is responsible for the small overhead in computation time we report.



(a) Without parallelization

(b) Only use CPU multithreading

(c) Use CUDA streams

Fig. 5: Result of GPU stream timeline from nvidia visual profile, MobileNet forward pass with 128 batch size

4.3 Result

Table 2 and table 3 show the performance of training VGG, ALexNet and MobileNet with 16-bit type of weights using our algorithm.

Table 2: Reduce memory usage by using our algorithm training neural networks

Batch Size	16	32	64	128	256
VGG	18.4%	25.0%	29.6%	-	-
AlexNet	6.5%	11.7%	20.3%	26.4%	27.5%
MobileNet	65.6%	66.8%	67.9%	68.4%	-

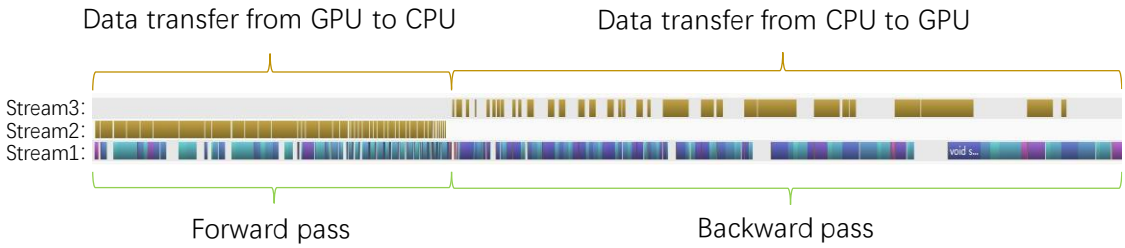


Fig. 6: GPU stream timeline of training MobileNet with batch size 128

Table. 3: Result of time increasing using our algorithm training neural networks

Batch Size		16	32	64	128	256
VGG	CPU multithreading	1.19x	1.23x	1.18x	-	-
	GPU stream	1.01x	1.01x	1.04x	-	-
AlexNet	CPU multithreading	1.17x	1.25x	1.20x	1.25x	1.26x
	GPU stream	1.0x	1.0x	1.02x	1.03x	1.06x
MobileNet	CPU multithreading	1.41x	1.46x	1.35x	1.32x	-
	GPU stream	1.21x	1.19x	1.18x	1.14x	-

As we can see from the results, we can dramatically reduce memory usage when training neural networks. Table 2, indicates that further memory reduction can be gained with larger batch sizes.

But MobileNet memory is significantly reduced compared to VGG and AlexNet because max pooling is used in both VGG and AlexNet, which adds additional memory consumption.

It can be seen from the table 3 that use more GPU stream to parallel training neural network is significantly efficient than only using CPU multithreading.

However, our algorithm depends on the cooperation between GPU computing speed and motherboard PCI-E transfer speed. For instance, when we training neural networks the computation is faster than data transfer in forward pass, we have to waiting data completely transfer into CPU memory. How to select the appropriate hardware to implement our algorithm is a problem needs to be considered.

5. Conclusion

Training convolution neural networks require high computing and large memory usage that is hard to implement in low performance device. We propose a general algorithm to reduce memory usage training neural networks by put input activation of each layers into CPU memory with using more GPU stream to parallel computing and data transfer. The result show that without much affecting speed of training, memory usage can be dramatically reduced.

References

[1] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems (pp. 1097-1105).

[2] Shallue, C. J., Lee, J., Antognini, J., Sohl-Dickstein, J., Frostig, R., & Dahl, G. E. (2018). Measuring the effects of data parallelism on neural network training. arXiv preprint arXiv:1811.03600.

[3] Martens, J., & Sutskever, I. (2012). Training deep and recurrent networks with hessian-free optimization. In Neural networks: Tricks of the trade (pp. 479-535). Springer, Berlin, Heidelberg.

[4] Gomez, A. N., Ren, M., Urtasun, R., & Grosse, R. B. (2017). The reversible residual network: Backpropagation without storing activations. In Advances in neural information processing systems (pp. 2214-2224).

[5] Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., ... & Adam, H. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861.

[6] Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., & Bengio, Y. (2017). Quantized neural networks: Training neural networks with low precision weights and activations. The Journal of Machine Learning Research, 18(1), 6869-6898.