Layer-Wise Invertibility for Extreme Memory Cost Reduction of CNN Training

Tristan Hascoet *
Kobe University
tristan@people.kobe-u.ac.jp

Quentin Febvre *
Sicara
quentin.febvre@gmail.com

Weihao Zhuang
Kobe University

Yasuo Ariki
Kobe University

Tetusya Takiguchi
Kobe University

Abstract

Convolutional Neural Networks (CNN) have demonstrated state-of-the-art results on various computer vision problems. However, training CNNs require specialized GPU with large memory. GPU memory has been a major bottleneck of the CNN training procedure, limiting the size of both inputs and model architectures. Given the ubiquity of CNN in computer vision, optimizing the memory consumption of CNN training would have wide spread practical benefits. Recently, reversible neural networks have been proposed to alleviate this memory bottleneck by recomputing hidden activations through inverse operations during the backward pass of the backpropagation algorithm. In this paper, we push this idea to extreme and design a reversible neural network with minimal training memory consumption. The result demonstrated that we can train CIFAR10 dataset on Nvidia GTX750 GPU only with 1GB memory and achieve 93% accuracy within 67 minutes.

1. Introduction

CNN have been successfully applied to many different computer vision applications. However, training state-of-the-art CNN requires special hardware with large memory capacity, as typical desktop GPU memory is too small for backpropagation training. Therefore, training deep networks comes with the barrier entry cost of either purchasing specialized hardware or renting real-time instances from cloud service providers. Reducing the memory consumption would allow to train neural networks efficiently in either on standard descktop GPUs or embedded devices. On-device training would also allow fine-tuning neural network models on local data without sending sensitive data over the network. Furthermore, a number of recent works have demonstrated the benefits of large batch training [1]. For example, linear speed-ups have been observed in training in Imagenet with batch sizes up to tens of thousands of samples. 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 [2] only store a fraction of the hidden activations and reconstructing the missing activations from the stored ones during the backward pass.

Reversible Network (RevNet) [3] constrain the architecture of Residual Networks to invertible transformations so that each layers input activations are reconstructed from their output during the backward pass. However, two factors create memory bottlenecks in training RevNet, the gradient of activations have to wait for the full reversible block recomputation that is local bottleneck. RevNet features non-volume preserving max-pooling layers, for which the inverse cannot be computed, their input must stored in memory that we call global bottleneck.

The iRevNet [4] model builds on the RevNet architecture: they replace the irreversible max-pooling with an invertible operation. One downside of their method is that the proposed invertible pooling scheme drastically increases the number of channels in upper layers. As the size of the convolution kernel weights grows quadratically, the memory cost associated with storing the model weights becomes a new memory bottleneck.

In this paper, we use a ResNet-18 as a starting point to analyse the training memory requirement of different invertible designs. We then introduce a layer-wise invertible architecture and characterize the accumulation of numerical errors across layers, which leads to numerical instabilities impacting model accuracy. We propose a hybrid architecture consist of layer-wise and residual-block-wise invertible operations that allow us to efficiently train CNN model by circumvent those bottleneck. The result show that training our hybrid model could achieve 93.3% accuracy on CIFAR10 dataset in 67 minutes on low-end Nvidia GTX750

*Equal contribution

GPU only with 1GB memory.

2. Memory footprint

We denote the memory footprint of training a neural network as a value $\mathcal{M}$ in bytes. The memory footprint represents the peak memory consumption during an iteration of training forward and backward pass. The total memory footprint $\mathcal{M} = \mathcal{M}_\theta + \mathcal{M}_z + \mathcal{M}_\delta$ consist of the sum of model weights, hidden activations and gradients.

The vanilla ResNet-18 do not use reversible computations so that the input activations need to be accumulated in memory during the forward pass for the computation of the weight gradients during in backward pass. Hence the peak memory footprint of training a vanilla ResNet happens at the beginning of the backward pass. Training this model on a batch of 32 RGB image of 240×240 requires 12.5 MB to store the model weights, and 3.8 GB of hidden layers activation and gradient, for a total of $\mathcal{M}$ = 3.81 GB. The memory cost of the hidden activations is thus the main memory bottleneck of CNN training.

Reversible blocks have analytical inverses that allow recomputation of both their input and hidden activations. The peak memory consumption of a RevNet, happens in the backward pass through the first reversible block. Following our previous example, a RevNet closely mimicking the ResNet-18 requires $\mathcal{M} = 1.19$ GB and 12.7 MB weights.

The iRevNet is fully invertible as it replaces max-pooling layers with invertible pooling layers. But, it requires 171 MB tostore its wights and $\mathcal{M}$ = 1.35 GB.

3. Methods

The memory bottleneck of RevNet and i-RevNet comes from the accumulation of hidden activations within a reversible block, upon their inverse computation during the backward pass before their gradient computation. In order to circumvent this local memory bottleneck, we aim to minimize the size of revertible blocks. In the extreme case, each revertible block consists of a unique layer: these are layer-wise invertible operations. As we shall shortly dscuss, these invertible operations introduce numerical error. Hence, we propose a hybrid model combining layer-wise and block-wise reversible operation to resolve the local memory bottleneck at the cost of a small additional computation cost. The following section describes such invertible layers, and, in Section 3.2, we introduce the hybrid architecture.

3.1. Layer-wise invertibility

Invertible batch normalization As batch normalization is not a bijective operation, it does not admit an analytical inverse. However, the input can be recomputed from the output at the minimal memory cost of saving the channel-wise mean and standard derivation statistics of the input batch.

We characterize the factor α of reduction of signal to noise ratio (SNR) through the inverse reconstruction. Let us consider a toy layer with only two channels and parameters $\beta = [0, 0]$ and $\gamma = [1, \rho]$ of first and second order moment parameters β and γ. For simplicity, let us consider an input signal x independently and identically distributed across both channels with zero mean and standard deviation 1 so that, in the forward pass, we have:

$$\alpha = \frac{snr(x)}{snr(y)} \tag{1a}$$

$$\alpha = \frac{||x||^2}{||\epsilon_x||^2} \frac{||\epsilon_y||^2}{||y||^2} \tag{1b}$$

$$\alpha = \frac{4}{(1 + \frac{1}{\rho^2}) \times (1 + \rho^2)} \tag{1c}$$

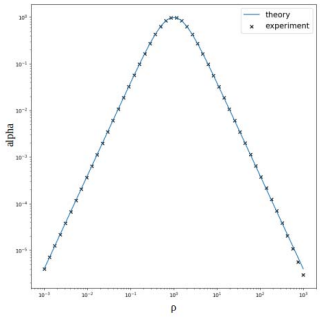

Figure 1. Illustration of the numerical errors arising from batch normalization layers.

Figure 1 shows the expected evolution of α through our toy layer for different values of the factor ρ. To validate our formula, we empirically evaluate α for normal Gaussian inputs x and output noise ϵ^y and find it to closely match the theoretical results given by equation 1.

Invertible activation function A good invertible activation function must be bijective (to guarantee the existence of an inverse function), and non-saturating (to prevent numerical errors). For these properties, we focus our attention on Leaky ReLUs. The analysis of the numerical errors yielded by the invertible Leaky ReLU follows a similar reasoning as the toy batch normalization example. We can think of the leaky ReLU as artificially splitting the input x across two different channels, one channel leaving the output unchanged and one channel that divides the input by a factor n during the forward pass and multiplies its output by a factor n during the backward pass. The signal to noise ration reduction factor α can be expressed as:

$$\alpha = \frac{4}{(1 + \frac{1}{n^2}) \times (1 + n^2)} \tag{2}$$

Hence numerical errors can be controlled by setting the value of the negative slope n, similarly to the parameter ρ in the toy batch normalization example.

Pooling In iRevNet, the authors propose an invertible pooling operation that operates by stacking the neighboring elements of the pooling regions along the channel dimension. We propose a new pooling layer that stacks the elements of neighboring pooling regions along the batch size instead of the channel size to circumvent quadratic increase in weight memory. We refer to both kind of pooling as channel pooling $\mathcal{P}_c$ and batch pooling $\mathcal{P}_b$ respectively. The reshaping operation performed by both pooling layers can be formalized as follows:

$$\mathcal{P}_c : \mathbb{R}^{bs \times c \times h \times w} \rightarrow \mathbb{R}^{bs \times 4c \times \frac{h}{2} \times \frac{w}{2}} \tag{3a}$$

$$\mathcal{P}_b : \mathbb{R}^{bs \times c \times h \times w} \rightarrow \mathbb{R}^{4bs \times c \times \frac{h}{2} \times \frac{w}{2}} \tag{3b}$$

The bs refers to the batch size, c to the number of channels and $h \times w$ to the resolution.

Invertible convolutions The inverse of a convolution is called deconvolution. However, deconvolution is computationally expensive and prone to numerical errors. We choose to implement invertible convolutions using the channel partitioning scheme as the reversible block design. Hence, in our architecture, invertible convolutions, can be seen as minimal reversible blocks in which both modules consist of a single convolution. Gomez *et al.* [3] found the numerical errors introduced by reversible blocks to have no impact on the model accuracy.

Full architecture Putting together the above building blocks. The peak memory usage for training iteration of this architecture requires $\mathcal{M}$ = 590 MB including 29.6 MB gradients. Similar to RevNet, the recomputation of activations during the backward pass comes with an additional computational cost similar to a forward pass.

In Figure 3, we compare the evolution of the accuracy in both settings for different depth and negative slopes. For small depths (or high negative slopes), in which the numerical errors are minimum, both models yield similar accuracy.

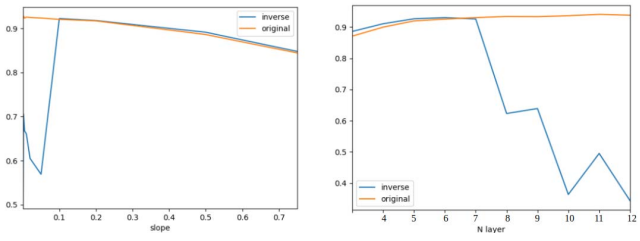

Figure 2. Impact of the numerical errors on the accuracy of layer-wise invertible models.

As the number of layer N increases (resp. the negative slope n decreases), the numerical errors increase, until they reach a threshold above which training diverges.

3.2. Hybrid architecture

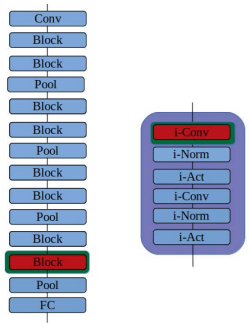

Figure 3. Illustration of a hybrid architecture and its peak memory consumption.

Through long chains of layer-wise invertible operations will accumulate numerical errors and show that numerical errors negatively impact model accuracy. To prevent these numerical instabilities, we introduce a hybrid architecture. Figure 3 illustrate our hybrid architecture, combining reversible residual blocks with layer-wise invertible function. Conceptually, the role of the layer-wise invertible layers is to efficiently recompute the hidden activations within the reversible residual blocks at the same time as the gradient propagates to circumvent the local memory bottleneck of the reversible module.

Figure 4 illustrate the backward pass though the hybrid reversible block. The backward pass is made of two phases: First the input activations are recomputed from the output using the Reversible block analytical inverse (middle). During this step, hidden activations are not kept in live memory so as to avoid the local memory bottleneck of the reversible block. Once the input activation recomputed, the gradients are propagated backward through both modules of the reversible blocks (right). During this second phase, hidden activations are recomputed backward through each module using the layer-wise inverse operations, yielding minimal memory footprint.

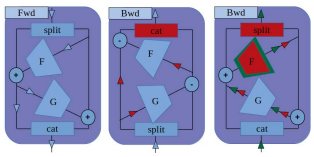

Figure 4. Illustration of the backpropagation process through a reversible block of our proposed hybrid architecture.

As each layer within these modules is invertible, the hidden activations are computed using the layer-wise inverse along the gradient. The layer-wise inversion allows us to al-

Table 1. Summary of architectures with different levels of reversibility

Model	Accuracy	#Params	Channels	Pooling	$\mathcal{M}$
Resnet	94.7%	$3.1M$	$32-64-128-256$	Max Pooling	$1.01G$
RevNet	94.5%	$3.1M$	$40-80-256-320$	Max Pooling	$348M$
i-RevNet	93.8%	$42.8M$	$32-128-512-2048$	$\mathcal{P}_c-\mathcal{P}_c-\mathcal{P}_c$	$500M$
Ours	93.3%	$3.7M$	$32-128-512-512$	$[\mathcal{P}_c,\mathcal{P}_c,\mathcal{P}_b]$	$200M$

leviate the local bottleneck of the reversible block by computing the hidden activation values together with the backward flow of the gradients. The peak memory consumption of our proposed architecture, training an iteration over batch of 32 images of resolution 240×240 would require $\mathcal{M}$ = 648MB including 14.8 MB weights.

It should be noted, instead of one additional forward pass, as in the RevNet and layer-wise architectures, our hybrid architecture comes with a computational cost equivalent to performing two additional forward passes during the backward pass.

4. Results

We use the CIFAR10 dataset as a benchmark for our experiments. The CIFAR10 dataset is complex enough to require efficient architectures to reach high accuracy, yet small enough to enable us to rapidly iterate over different architectural designs.

We summarize the benefits and drawbacks of our proposed architecture in comparison to different baseline architectures. Table 1 summarizes our main results. In this table, we compare architectures with different patterns of reversibility. To allow for a fair comparison, we have tweaked each architecture to keep the number of parameters as close as possible.

All models were trained for 50 epochs of stochastic gradient descent with cyclical learning rate and momentum with minimal image augmentation.

Compared to the original ResNet architecture, our model drastically cuts the memory cost of training. These drastic memory cuts come at the cost of a small degradation in accuracy.

Furthermore, our hybrid architecture requires two equivalent additional forward computation within each backward pass. In Table 2, we compare the time of training our hybrid architecture to 93.3% accuracy on Nvidia GTX1080Ti and low-end Nvidia GTX750 GPU which only has 1 GB memory and roughly 400 MB of available memory after the initialization of various frameworks. It is impractical train a vanilla ResNet with large batch size, while our architecture allows for efficient training.

Table 2. Training statistics on different hardware

GPU	Accuracy	Time
GTX750	93.3%	$67min$
GTX 1080Ti	93.3%	$37min$

5. Conclusion

Convolutional Neural Networks have become the backbone of computer vision systems. Despite their great success, one major drawback of these models is their intense resource consumption: Training CNNs needs highly optimized implementations leveraging all possible hardware resources. In this paper, we have presented an architecture able to yield high accuracy classifications within very tight memory constraints and train a hybrid CNN model to 93.3% accuracy on a low-end GPU only 1 GB memory.

References

[1] S. McCandlish, J. Kaplan, D. Amodei, and O. Dota Team, "An empirical model of large-batch training," *arXiv preprint arXiv:1812.06162*, 2018. 1

[2] T. Chen, B. Xu, C. Zhang, and C. Guestrin, "Training deep nets with sublinear memory cost," *arXiv preprint arXiv:1604.06174*, 2016. 1

[3] A. N. Gomez, M. Ren, R. Urtasun, and R. B. Grosse, "The reversible residual network: Backpropagation without storing activations," in *Advances in Neural Information Processing Systems*, pp. 2214–2224, 2017. 1, 3

[4] J.-H. Jacobsen, A. Smeulders, and E. Oyallon, "i-revnet: Deep invertible networks," *arXiv preprint arXiv:1802.07088*, 2018. 1