Deep Learning Approximation: Zero-Shot Neural Network Speedup

Anonymous Author(s)
Affiliation
Address
email

Abstract

Neural networks offer high-accuracy solutions to a range of problems, but are computationally costly to run in production systems. We propose a technique called Deep Learning Approximation to take an already-trained neural network model and build a faster (and almost equally accurate) network by manipulating the network structure and coefficients without requiring re-training or access to the training data. Speedup is achieved by applying a sequential series of independent optimizations that reduce the floating-point operations (FLOPs) required to perform a forward pass. An optimal lossy approximation is chosen for each layer by weighing the relative accuracy loss and FLOP reduction. On PASCAL VOC 2007 with the YOLO network, we show an end-to-end 2x speedup in a network forward pass with a 5% drop in mAP that can be re-gained by finetuning, enabling this network (and others like it) to be deployed in compute-constrained systems.

1 Introduction

At deploy time, the dollar cost of a production pipeline using a neural network is proportional to the time required to execute a forward pass, which is proportional to the number of floating-point operations (FLOPs) required. We want to run faster models to achieve higher throughput, lower cost, better hardware utilization, and lower power, cooling, and CPU speed requirements. However, training the smallest possible network to achieve the desired task is challenging.

Instead, we propose the DLA method to take a network that is slightly too slow and reduce the FLOP count with minimal accuracy loss, reducing the need to train and re-train to find an appropriately-sized network. We automatically select an appropriate singular value decomposition (SVD) to apply to each weight tensor in the original network, replacing it with a set of weights that, when applied sequentially, offer similar outputs but using fewer FLOPs. Intuitively, using low-rank decomposition to determine which FLOPs to keep and which to approximate means that representation layers that are redundant will have a lower-rank decomposition and yield more speedup for less accuracy loss when approximated. The sensitivity of this approximation is controlled by a single parameter that represents whether accuracy or speedup is more important. Because this method does not require access to the original network training data, it can be used to speed up systems where networks are a black-box or training data is confidential.

On a benchmark set of standard computer vision tasks and neural network architectures, we show between a 1.5x and 2x speedup without additional re-training and minimal accuracy loss. In each case, funetuning after applying DLA recovers lost accuracy.

This work extends that of Denton et al. [1], who use SVD and factorization approaches to approximate weight layers, but rely on re-training after every layer has been approximated. DLA extends this method by applying approximations in a zero-shot manner, without requiring re-training.

Submitted to Workshop on Compact Deep Neural Network Representation with Industrial Applications at 32nd Conference on Neural Information Processing Systems (NIPS 2018). Do not distribute.

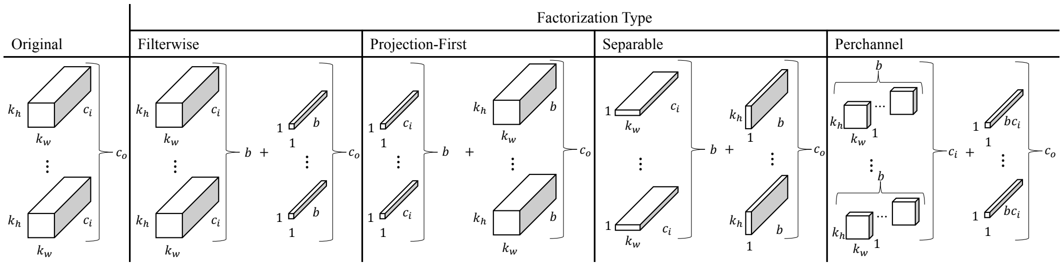

Figure 1: A visual representation of the 4 factorization methods used in DLA. c_i is the number of input channels, c_o is the number of output channels, k_w is the kernel width, k_h is the kernel height, and b is the factorization parameter.

2 Computationl Model and Weight Tensors

According to the roofline computation model [11], network runtime is dominated by FLOP count and memory movement to and from the GPU. Weight layers fall in the FLOP-dominated regime because they have a high arithmetic intensity. DLA replaces FLOP-heavy operations with operations that have a lower FLOP count, pushing the layers' operations into the memory-limited regime.

The FLOP count is dominated by the duplication factor on the input blob size. A convolutional layer is a 4D tensor W in $\mathbb{R}^{c_o \times c_i \times k_h \times k_w}$, where c_o is the number of output feature maps, c_i is the number of input channels, and k_h and k_w are the kernel dimensions in y and x respectively. The convolution traverses an input blob in $\mathbb{R}^{h \times w}$ with stride s_h in y and s_w in x. In a grouped convolution, the input and output channels are broken into g groups and the 4D weight tensor is in $\mathbb{R}^{c_o \times \frac{c_i}{g} \times k_h \times k_w}$.

3 The Approximation Pipeline

Table 1: Properties of 4 types of SVD approximations, shown as a multiple of hw

	Filterwise	**Projection-First**	**Separable**	**Perchannel**
W_1 shape W_2 shape	$\mathbb{R}^{b \times c_i \times k_h \times k_w}$ $\mathbb{R}^{c_o \times b \times 1 \times 1}$	$\mathbb{R}^{b \times c_i \times 1 \times 1}$ $\mathbb{R}^{c_o \times b \times k_h \times k_w}$	$\mathbb{R}^{b \times c_i \times k_h \times 1}$ $\mathbb{R}^{c_o \times b \times 1 \times k_w}$	$\mathbb{R}^{bc_o \times \frac{c_i}{c_o} \times k_h \times k_w}$ $\mathbb{R}^{c_o \times bc_o \times 1 \times 1}$
FLOPs for $W_1 + W_2$	$\frac{c_i b k_h k_w}{s_h s_w g} + \frac{bc_o}{s_h s_w}$	$c_i b + \frac{bc_o k_h k_w}{s_h s_w g}$	$\frac{c_i b k_h}{s_h s_w g} + \frac{bc_o k_w}{s_h s_w}$	$\frac{c_i b k_h k_w}{s_h s_w} + \frac{b{c_o}^2}{s_h s_w}$
A	$\frac{{s_b}^2}{\sum_{j=0}^{c_o} {s_j}^2}$	$\frac{{s_b}^2}{\sum_{j=0}^{c_o} {s_j}^2}$	$\frac{{s_b}^2}{\sum_{j=0}^{c_o} {s_j}^2}$	$\frac{1}{c_i} \sum_{c=0}^{c_i} \frac{{s_{c,b}}^2}{\sum_{j=0}^{c_o} {s_{c,j}}^2}$
R	$\frac{b}{c_o} + \frac{bg}{c_i k_h k_w}$	$\frac{b s_h s_w g}{c_o k_w k_h} + \frac{b}{c_i}$	$\frac{b}{c_o k_w} + \frac{bg}{c_i k_h}$	$\frac{bg}{c_o} + \frac{bgc_o}{c_i k_w k_h}$

An optimal approximation is chosen by calculating the runtime and accuracy loss from all possible decompositions (including chaining multiple approximations) and selecting the one with the highest score. The lossy approximations that are enumerated (both independently and chained together) are low-rank approximations of the original weight tensor W using SVD [4], resulting in two convolutions W_1 and W_2 that can be applied sequentially in place of W, shown visually in Figure 1. An **accuracy score** A (the percentage of variation explained by the approximation) and **runtime score** R (the FLOP reduction) is computed for each approximation. The operation with the highest combined score $pA + (1-p)R$ is selected. A table of SVD decompositions and FLOP reductions for each type of approximation is shown in Table 1. When chaining approximations, R for is the ratio of the final output FLOPs to the FLOPs from W. A is the product of the accuracy scores for each approximation in the chain, since any error introduced by the first will be carried over to the next.

Table 2: DLA applied to 4 networks and datasets

	Runtime (ms)			Accuracy (top-1 or mAP)		
Network / dataset	**Baseline**	**DLA**	**Speedup**	**Baseline**	**DLA**	**Finetuned**
AlexNet [7] / CIFAR10 [6]	6.06	3.47	1.75x	70.3%	67.75%	68.8%
ResNet50 [3] / ImageNet2012 [9]	40.9	26.8	1.50x	72.3%	62.2%	68.6%
VGG16 [10] / ImageNet2012	78.6	45.2	1.75x	70.38%	59.60%	70.4%
YOLO [8] / VOC2007 [2]	63	31	2.00x	66.9	61.9	65.9

4 Experimental results

Table 2 shows between 1.5x and 2x runtime improvement, between 5 and 10% loss in accuracy, and nearly full recovery of accuracy after finetuning on 4 standard networks.[1] FLOP reduction correlates with the absolute speedup, with the exception of the ResNet50 network, as shown in Table 3. Table 4 shows that the input parameter p can be chosen based on the desired runtime / accuracy tradeoff.

Table 3: Speedup, FLOP reduction, and memory reduction

Network on dataset	**Speedup**	**FLOP ↓**	**Memory ↓**
AlexNet [7] on CIFAR10 [6]	1.75x	2.50x	1.10x
ResNet50 [3] on ImageNet2012 [9]	1.50x	1.20x	1.50x
VGG16 [10] on ImageNet2012	1.75x	1.70x	1.50x
YOLO [8] on VOC2007 [2]	2.00x	2.00x	1.60x

Table 4: Runtime and accuracy on YOLO

p	**ms**	**mAP**
baseline	63	66.9
0.9	57	66.9
0.8	54	66.0
0.7	45	65.4
0.6	41	65.0
0.5	32	61.9
0.4	27	50.0

Memory-limited layers such as fully-connected layers and 1×1 convolutions do not see as much of an improvement in runtime from DLA (for example ResNet50 in Table 3). Additionally, pushing beyond the 2x speedup observed on YOLO without significant accuracy loss is not possible with DLA, because layers are moved from the FLOP-limited regime to the memory-limited regime so more aggressive approximation trades off mode accuracy for less runtime improvement.

5 Conclusion

After networks see diminishing returns with FLOP-reduction methods like DLA, it means that memory movement is the runtime bottleneck, and memory-reducing optimizations should be applied, suggesting a future research direction into zero-shot memory reduction.

Deep Learning Approximation can be applied to an already-trained network to speed it up and incur only a small amount of accuracy loss. Access to training data is not required and the techniques are framework-agnostic, which means DLA can be used on black-box networks or in environments where the training data cannot be accessed. The combination of approximations that best achieve the desired accuracy loss is chosen for each layer through an exhaustive search. DLA can be combined with other methods for speeding up neural networks. This runtime reduction can generate a multiplier in cost reduction for production services that use neural networks. Any accuracy loss that that was introduced can be recovered by fine-tuning or re-training the new resultant network.

[1] Runtime was tested on a single input with Caffe [5] compiled with CUDA8 on a g2 cloud instance.

References

[1] Emily Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 1*, pages 1269–1277, 2014.

[2] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, 2007. URL `http://link.springer.com/10.1007/s11263-009-0275-4`.

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[4] M. Jaderberg, A. Vedaldi, and A. Zisserman. Speeding up convolutional neural networks with low rank expansions. In *Proceedings of the British Machine Vision Conference (BMVC)*, 2014.

[5] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. 2014.

[6] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 (canadian institute for advanced research). Technical report. URL `http://www.cs.toronto.edu/~kriz/cifar.html`.

[7] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. 2012.

[8] Joseph Redmon and Ali Farhadi. Yolo9000: Better, faster, stronger. 2016.

[9] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. Imagenet large scale visual recognition challenge. *Int. J. Comput. Vision*, 10(1007):11263–015, December 2015.

[10] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. Technical report, 2014.

[11] Samuel Williams, Andrew Waterman, and David Patterson. Roofline: An insightful visual performance model for multicore architectures. In *Commun. ACM*, pages 65–76, April 2009.