ON IMPROVING THE NUMERICAL STABILITY OF WINOGRAD CONVOLUTIONS

Kevin Vincent, Kevin Stephano, Michael Frumkin, Boris Ginsburg & Julien Demouth
NVIDIA
{kvincent,kstephano,mfrumkin,bginsburg,jdemouth}@nvidia.com

ABSTRACT

Deep convolutional neural networks rely on heavily optimized convolution algorithms. Winograd convolutions provide an efficient approach to performing such convolutions. Using larger Winograd convolution tiles, the convolution will become more efficient but less numerically accurate. Here we provide some approaches to mitigating this numerical inaccuracy. We will exemplify these approaches by working on a tile much larger than any previously documented: $F(9 \times 9, 5 \times 5)$. Using these approaches, we will show that such a tile can be used to train modern networks and provide performance benefits.

1 INTRODUCTION

Deep convolutional neural networks have become a foundation for modern computer vision. These are heavily compute intensive operations which consume days to weeks to train on GPUs. The majority of this time is spent in convolution layers which puts this layer at the top of our focus.

The most efficient implementation of the convolutional layer is based on Winograd Lavin & Gray (2015). Winograd convolutions work on tiles designed for a specific output size and filter size. While larger output & filter size tiles provide better complexity reduction, they also provide higher numerical instability. Following Lavin & Gray (2015), we will refer to tile sizes by $F(m \times m, n \times n)$ where m is the number of outputs and n is the number of weights. The input size of a tile is computed as $m + n - 1 \times m + n - 1$.

There has been little investigation on moving beyond $F(4 \times 4, 3 \times 3)$ tiles; the assumption being they are too inaccurate. To demonstrate very large tiles can perform accurate computations, we will exemplify a tile with both a larger output & filter size: $F(9 \times 9, 5 \times 5)$.

We investigated the source of numerical instability for Winograd convolutions. We propose two approaches to mitigate this instability; we will demonstrate these approaches with $F(9 \times 9, 5 \times 5)$ as our example. We will show improvements to numerical stability and empirically prove that modern networks can train successfully while using the $F(9 \times 9, 5 \times 5)$ tile; Alexnet Krizhevsky et al. (2012) and Inception v3 Szegedy et al. (2015) as our examples. We will also show performance improvements, comparing to cuDNN v6, for the 5×5 convolution layers in these networks.

2 GENERATION OF WINOGRAD TRANSFORM MATRICES USING VANDERMONDE

A convolution correlates K filters with C channels of size $R \times S$ against a minibatch of N images with C channels of size $H \times W$. We will name image elements as $x_{n,c,h,w}$ and filter elements as $w_{k,c,r,s}$.

A single output $y_{n,k,p,q}$ can be computed via the following:

$$y_{n,k,p,q} = \sum_{c=1}^{C} \sum_{r=1}^{R} \sum_{s=1}^{S} x_{n,c,p+r,q+s} w_{k,c,r,s} \tag{1}$$

From Lavin & Gray (2015), the outputs of convolution can be computed efficiently via the following:

$$x * w = Y^T(X^T x X \bigodot W w W^T)Y \tag{2}$$

For a tile of size $F(m \times m, n \times n)$, the following is one method for computing these efficient transform matrices (Y_T, X_T & W).

$$Y^T = (V_{(m+n-1)\times m})^T S_Y \tag{3}$$

$$X^T = S_X(V_{(m+n-1)\times(m+n-1)})^{-T} \tag{4}$$

$$W = S_W(V_{(m+n-1)\times n}) \tag{5}$$

Where $V_{a\times b}$ is a trimmed Vandermonde matrix for Homogenous Coordinate polynomials. This can be expressed directly with the following:

$$\begin{bmatrix} {f_0}^0{g_0}^{b-1} & {f_0}^1{g_0}^{b-2} & \cdots & {f_0}^{b-1}{g_0}^0 \\ {f_1}^0{g_1}^{b-1} & {f_1}^1{g_1}^{b-2} & \cdots & {f_1}^{b-1}{g_1}^0 \\ \vdots & & & \vdots \\ {f_{a-1}}^0{g_{a-1}}^{b-1} & {f_{a-1}}^1{g_{a-1}}^{b-2} & \cdots & {f_{a-1}}^{b-1}{g_{a-1}}^0 \end{bmatrix} \tag{6}$$

Where (f_i, g_i) are given unique homogenous coordinates (we will call polynomial points). Note that the chosen (f_i, g_i) must be the same provided for all matrices (Y^T, X^T and W).

S_Y, S_X and S_W are given diagonal square matrices where $S_Y S_X S_W = I$. When referring to these matrices, we will only list the diagonal values as a vector; these values follow the diagonal square matrix from left-right.

The transforms provided by Lavin & Gray (2015) can be generated using this generation method with the following polynomial points and scalings.

$F(2 \times 2, 3 \times 3)$ & $F(3 \times 3, 2 \times 2)$

Polynomial Points	[(0, 1), (1, 1), (-1, 1), (1, 0)]
S_Y	[1, 1, 1, -1]
S_X	[1, 2, 2, -1]
S_W	[1, $\frac{1}{2}$, $\frac{1}{2}$, 1]

$F(4 \times 4, 3 \times 3)$

Polynomial Points	[(0, 1), (1, 1), (-1, 1), (2, 1), (-2, 1), (1, 0)]
S_Y	[1, 1, 1, 1, 1, 1]
S_X	[4, -6, -6, 24, 24, 1]
S_W	[$\frac{1}{4}$, -$\frac{1}{6}$, -$\frac{1}{6}$, $\frac{1}{24}$, $\frac{1}{24}$, 1]

Note that Fourier Transform is a special case of the Vandermonde matrix. This special case requires a complex-space Vandermonde matrix with polynomial points being roots of unity. As Winograd convolutions are conventionally in real-space, we will ignore complex-space Vandermonde matrices for further discussions.

3 Approaches to mitigate instability of Vandermonde matrices

Vandermonde matrices tend to be numerically unstable for large sizes Pan (2015). This is a direct consequence of the exponential growth of polynomial points in Vandermonde matrices. Well chosen polynomial points can mitigate this instability as some values exhibit minimal exponential growth. There are several approaches to finding these minimal polynomial points but we have found the best success from a basic approach of minimizing the numerator & denominator. For example, we have found values such as $(2, 1)$ & $(\frac{1}{2}, 1)$ tend to cause less error than values such as $(3, 1)$ & $(\frac{1}{3}, 1)$.

Our special use case of Vandermonde matrices also provides another approach to mitigate instability. We can choose scaling matrices S_Y, S_X, S_W such that we can mitigate the exponential growth.

Using both of the above approaches, we have been able to successfully train Alexnet and Inception v3 using $F(9 \times 9, 5 \times 5)$ tiles for all 5×5 convolutions and $F(4 \times 4, 3 \times 3)$ for all 3×3 tiles; using direct convolution for all other layers.

For our $F(9 \times 9, 5 \times 5)$ convolutions, we choose the following polynomial points: [(0, 1), (1, 1), (-1, 1), ($\frac{1}{2}$, 1), ($-\frac{1}{2}$, 1), ($\frac{1}{3}$, 1), ($-\frac{1}{3}$, 1), ($\frac{3}{2}$, 1), ($-\frac{3}{2}$, 1), (-3, 1), (2, 1), (-2, 1), (1, 0)]. See Appendix A for the resulting output transform matrix for such polynomial points.

Observing the output transform matrix, we hand-pick the following scalings for S_Y: [-1.333333, 0.05, 0.1, -0.7314286, -1.024, 1.314635, 1.643293, -0.005277263, -0.01583179, -1.587302e-05, 0.0003265306, 0.001632653, 1]. See Appendix B for the result of applying such scalings.

For the $F(9 \times 9, 5 \times 5)$ tile, the Y^T matrix exhibits exponential growth up to power $9 - 1 = 8$ where as W matrix exhibits exponential growth up to $5 - 1 = 4$. Because W has significantly less exponential growth, we ignore scalings for W by setting S_W to the Identity matrix.

Once S_Y and S_W are known, there is only one such matrix S_X that will meet the requirement of $S_Y S_X S_Y = I$. As such, S_X must be [-0.75, 20, 10, -1.367188, -0.9765625, 0.7606677, 0.6085342, -189.4922, -63.16406, -63000, 3062.5, 612.5, 1].

These scalings reduce the matrix condition number for S_Y, S_X, S_W from [36279, 113237696, 64] to [125, 2094, 64]. Running numerical experiments on the 5×5 convolutions in Alexnet & Inception, the error has been reduced by 137x and 43x respectively; see following Section 4 for more info. Using these scalings, we have been able to successfully train Alexnet and Inception v3 using $F(9 \times 9, 5 \times 5)$ tiles for all 5×5 convolutions and $F(4 \times 4, 3 \times 3)$ for all 3×3 convolutions; using direct convolution for all other layers.

4 Experimental Results

We use a simple non-fused approach to implement $F(9 \times 9, 5 \times 5)$ where all transform matrices are computed in separate CUDA kernels and cuBLAS SGEMM is called to compute the point-wise elements of all N, C and K. All Winograd tiles are computed at the same time which means all tiles must be resident in GPU memory.

For performance experiments, we use an NVIDIA GTX Titan X (Pascal) and cuDNN v6. We provide the speedup from the best cuDNN v6 algorithm compared with $F(9 \times 9, 5 \times 5)$ compute time for forward convolution (conv), data gradient (dgrad) and weight gradient (wgrad) computation. Note that the $F(9 \times 9, 5 \times 5)$ algorithm is present in cuDNN v6, we compare this algorithm with with best of all other algorithms. We use a batch size of 32.

Layer	H/W	Pad	C	K	conv	dgrad	wgrad
Alexnet 5×5 layer	27	2	48	128	1.05x	1.23x	1.90x
Inception 5×5 layer	35	2	48	64	1.42x	1.44x	1.37x

For numerical experiments, reference is computed in fp64 with data & weights from the uniform distribution [-1, 1]. The error shown is the maximum relative error of all computed elements for a forward convolution; all using fp32 compute and storage. Direct being a direct convolution approach. Non-Scaled being $F(9 \times 9, 5 \times 5)$ without any scalings discussed in Section 3. Scaled being the previous but with the scalings. We use a batch size of 32.

Layer	H/W	Pad	C	K	Direct	Non-Scaled	Scaled
Alexnet 5×5 layer	27	2	48	128	2.81E-06	7.53E-02	5.49E-04
Inception 5×5 layer	35	2	48	64	2.84E-06	2.16E-02	4.98E-04

5 Future Work

Our results with $F(9 \times 9, 5 \times 5)$ strongly suggest that $F(9 \times 9, 3 \times 3)$ tiles will be accurate enough for convolutional neural networks. This large tile could provide speedups to 3×3 convolutions which are much more prevalent than 5×5.

Our results also indicate that approaches to mitigate instability can have profound impacts on the accuracy of Winograd convolutions. Further exploration could lead to further stability and even larger tiles.

REFERENCES

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks, 2012.

Andrew Lavin and Scott Gray. Fast algorithms for convolutional neural networks, 2015.

Victor Y. Pan. How bad are vandermonde matrices?, 2015.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision, 2015.

APPENDIX A NON-SCALED $V_{13\times9}$

$V_{13\times9}$ matrix (output transform) for polynomial points: [(0, 1), (1, 1), (-1, 1), ($\frac{1}{2}$, 1), ($-\frac{1}{2}$, 1), ($\frac{1}{3}$, 1), ($-\frac{1}{3}$, 1), ($\frac{3}{2}$, 1), ($-\frac{3}{2}$, 1), (-3, 1), (2, 1), (-2, 1), (1, 0)].

1	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1
1	-1	1	-1	1	-1	1	-1	1
1	0.5	0.25	0.125	0.0625	0.031	0.01562	0.0078	0.0039
1	-0.5	0.25	-0.125	0.0625	-0.031	0.01562	-0.0078	0.0039
1	0.3333	0.1111	0.037	0.01235	0.0041	0.00137	0.0004	0.00015
1	-0.3333	0.1111	-0.037	0.01235	-0.0041	0.00137	-0.0004	0.00015
1	1.5	2.25	3.375	5.062	7.594	11.39	17.09	25.63
1	-1.5	2.25	-3.375	5.062	-7.594	11.39	-17.09	25.63
1	-3	9	-27	81	-243	729	-2187	6561
1	2	4	8	16	32	64	128	256
1	-2	4	-8	16	-32	64	-128	256
0	0	0	0	0	0	0	0	1

APPENDIX B SCALED $V_{13\times9}$

$V_{13\times9}$ matrix (output transform) for polynomial points: [(0, 1), (1, 1), (-1, 1), ($\frac{1}{2}$, 1), ($-\frac{1}{2}$, 1), ($\frac{1}{3}$, 1), ($-\frac{1}{3}$, 1), ($\frac{3}{2}$, 1), ($-\frac{3}{2}$, 1), (-3, 1), (2, 1), (-2, 1), (1, 0)]. Each row is scaled by the following factors: [-1.333333, 0.05, 0.1, -0.7314286, -1.024, 1.314635, 1.643293, -0.005277263, -0.01583179, -1.587302e-05, 0.0003265306, 0.001632653, 1].

-1.3	0	0	0	0	0	0	0	0
0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05
0.1	-0.1	0.1	-0.1	0.1	-0.1	0.1	-0.1	0.1
-0.73	-0.37	-0.18	-0.091	-0.046	-0.023	-0.011	-0.0057	-0.0029
-1	0.51	-0.26	0.13	-0.064	0.032	-0.016	0.008	-0.004
1.3	0.44	0.15	0.049	0.016	0.0054	0.0018	0.0006	0.0002
1.6	-0.55	0.18	-0.061	0.02	-0.0068	0.0023	-0.0007	0.00025
-0.0053	-0.0079	-0.012	-0.018	-0.027	-0.04	-0.06	-0.09	-0.14
-0.016	0.024	-0.036	0.053	-0.08	0.12	-0.18	0.27	-0.41
-1.6e-05	4.8e-05	-0.0001	0.00043	-0.0013	0.0039	-0.012	0.035	-0.1
0.00033	0.00065	0.0013	0.0026	0.0052	0.01	0.021	0.042	0.084
0.0016	-0.0033	0.0065	-0.013	0.026	-0.052	0.1	-0.21	0.42
0	0	0	0	0	0	0	0	1