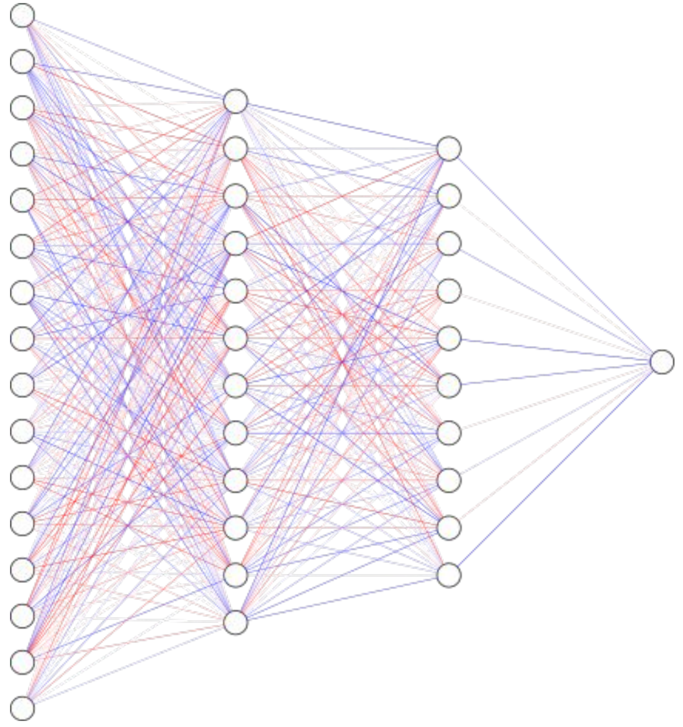
Applying Fast Matrix Multiplication to Neural Networks

Ahmed Khaled, Amir F. Atiya, Ahmed H. Abdel-Gawad
Cairo University, Egypt

The 35th ACM/SIGAPP Symposium On Applied Computing (ACM SAC), March-April 2020

Deep Neural Networks

- Large, compositional machine learning models.
- Ubiquitous in practice.
- We focus on *feedforward neural networks* with *fully-connected* layers.



Limitations of Training Neural Networks

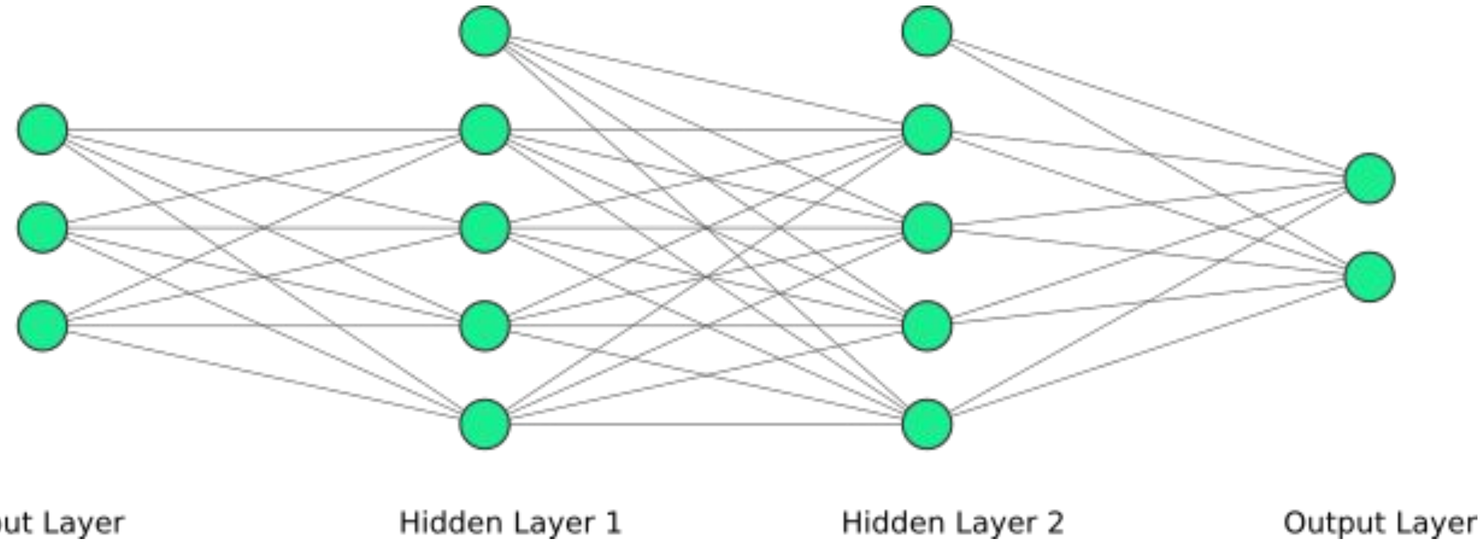
- Neural networks have been getting wider with many layers.
- It takes a lot of time and resources to train them. This is a burden on the practitioner.
- There are many ways to improve this:
    - Optimized computation kernels.
    - Start from pre-trained models.
    - Use approximate computations.

Our Contribution

- We apply a different matrix multiplication algorithm, *Winograd's Algorithm*, to training neural networks on GPUs using TensorFlow.
- We investigate the applicability and limitations of this method in practice.

Feedforward Neural Networks

- We assume a loss function L to be given and applied to the output.



Computation in a single layer

- Define

$$X = \begin{bmatrix} x_1^1 & x_2^1 & x_3^1 & \dots & x_K^1 \\ x_1^2 & x_2^2 & x_3^2 & \dots & x_K^2 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_1^N & x_2^N & x_3^N & \dots & x_K^N \end{bmatrix}.$$

Input matrix

$$B = \begin{bmatrix} b_1 & b_2 & b_3 & \dots & b_M \\ b_1 & b_2 & b_3 & \dots & b_M \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ b_1 & b_2 & b_3 & \dots & b_M \end{bmatrix}.$$

(Stacked) Bias vector

Computation in Feedforward Neural Networks

- Then the output of a feedforward neural network layer is

$\dim(Y) = N \times M$

Weight Matrix $\dim(W) = K \times M$

$$Y = \sigma(XW + B)$$

Activation Function $\sigma : \mathbb{R} \to \mathbb{R}$

Computation in Feedforward Neural Networks

- In order to use (stochastic) gradient descent, we apply backpropagation. The gradients for each layer:

Element-wise product

Propagated loss

$$\frac{\partial L}{\partial W} = X^T \left(\sigma'(XW + B) \odot \frac{\partial L}{\partial Y} \right),$$

$$\frac{\partial L}{\partial B} = \sigma'(XW + B) \odot \frac{\partial L}{\partial Y}.$$

Activation function derivative

Speeding up computation

- The matrix multiplication step dominates the computation complexity of both the forward and backward passes.
- Current implementations in popular linear algebra libraries (like Eigen or cuBLAS) rely on Goto's algorithm or variations of it, a refinement of the ordinary three-loop procedure.
- The three-loop algorithm has complexity $\mathcal{O}(n^3)$ which is not optimal.

Fast Matrix Multiplication

- There are alternative matrix multiplication algorithm that beat the cubic rate such as Strassen’s algorithm, Winograd’s algorithm, and Coppersmith–Winograd algorithm.
- Many are not practical (Dumas and Pan 2016). However, recent research has focused on practical implementations of fast matrix multiplication algorithms, including GPU implementations (Lei et al. 2013, Huang et al. 2018).

Winograd’s Algorithm

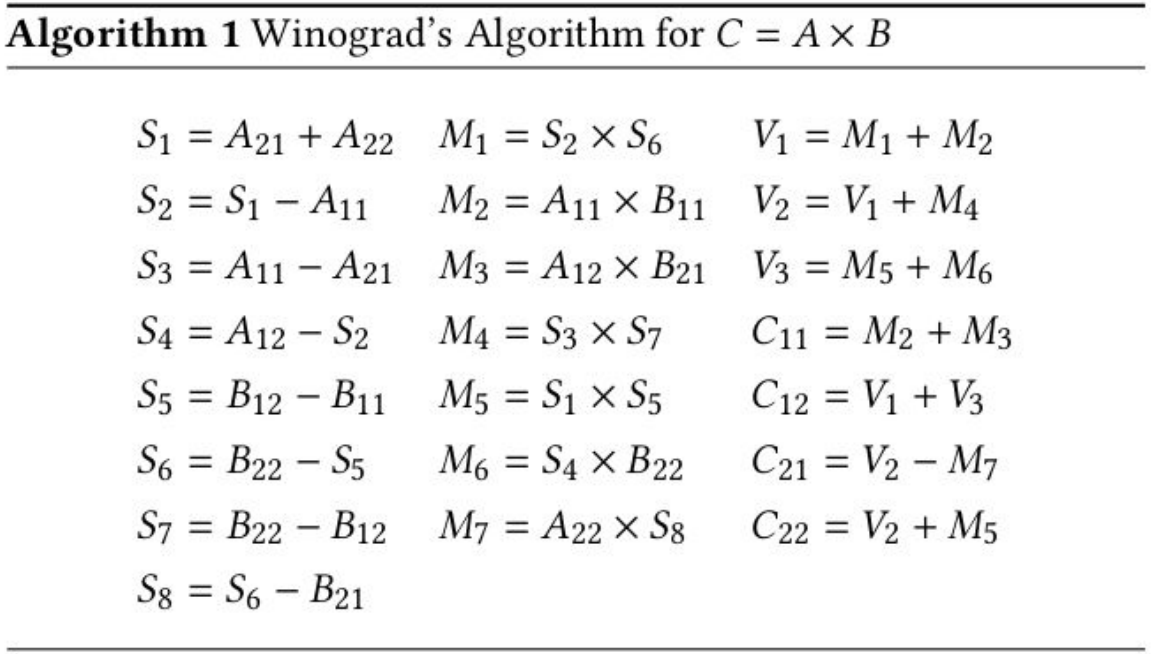
- If we want to compute $C = AB$, we decompose the matrices into blocks

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}.$$

- The straightforward way of multiplying the submatrices requires 8 sub-matrix multiplications and 4 additions.

Winograd's Algorithm

Winograd's Algorithm allows us to use seven multiplications and eighteen additions. This is asymptotically faster.



Algorithm 1 Winograd's Algorithm for $C = A \times B$

$S_1 = A_{21} + A_{22}$	$M_1 = S_2 \times S_6$	$V_1 = M_1 + M_2$
$S_2 = S_1 - A_{11}$	$M_2 = A_{11} \times B_{11}$	$V_2 = V_1 + M_4$
$S_3 = A_{11} - A_{21}$	$M_3 = A_{12} \times B_{21}$	$V_3 = M_5 + M_6$
$S_4 = A_{12} - S_2$	$M_4 = S_3 \times S_7$	$C_{11} = M_2 + M_3$
$S_5 = B_{12} - B_{11}$	$M_5 = S_1 \times S_5$	$C_{12} = V_1 + V_3$
$S_6 = B_{22} - S_5$	$M_6 = S_4 \times B_{22}$	$C_{21} = V_2 - M_7$
$S_7 = B_{22} - B_{12}$	$M_7 = A_{22} \times S_8$	$C_{22} = V_2 + M_5$
$S_8 = S_6 - B_{21}$		

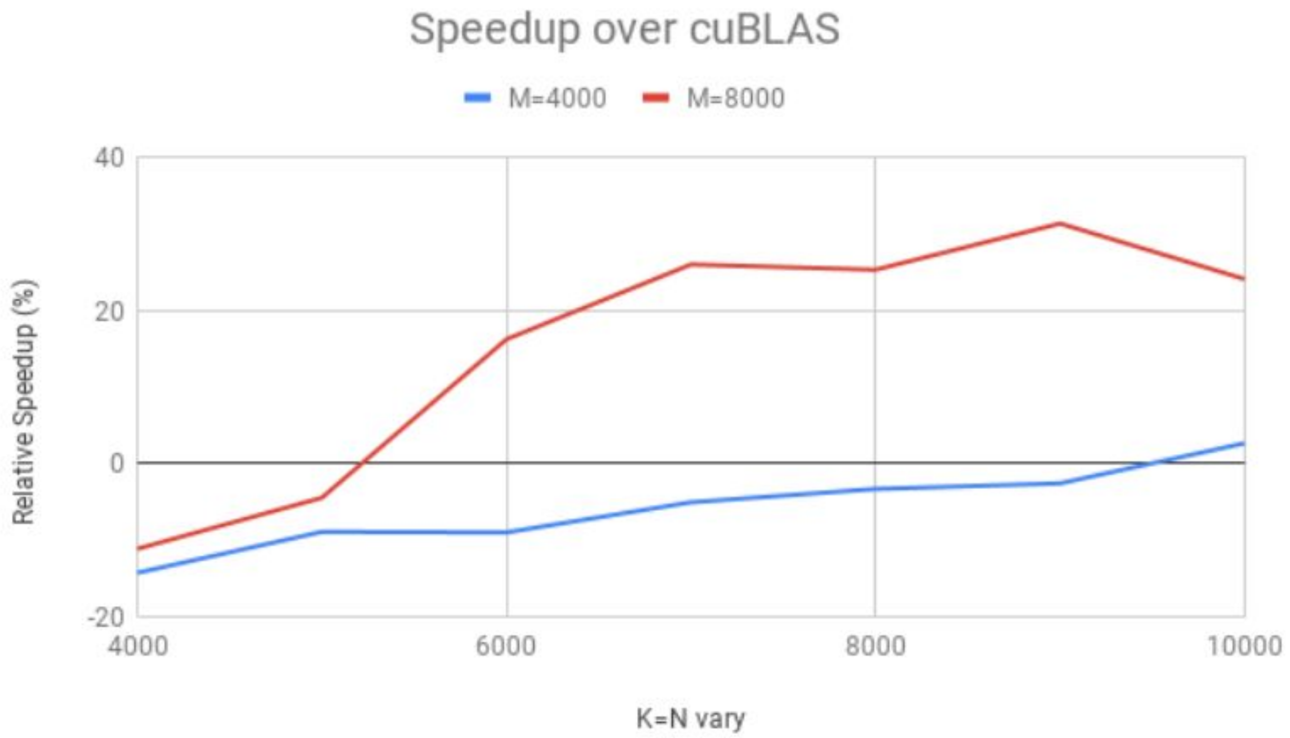
Practical Implementations of Winograd’s Algorithm

- The algorithm is not applied recursively to the scalar level, but usually only for one or two levels of recursion and then ordinary matrix multiplication is used.
- The previous formulation requires a lot of temporary memory for storing submatrices. Instead we use a different but equivalent formulation from (Lai et al. 2013) that requires storing only two temporary submatrices.

Application to Neural Networks

- We implement feedforward layers in TensorFlow and use Winograd's Algorithm for matrix multiplication. We test against cuBLAS, the standard library on nVidia GPUs.
- We report the result as we vary the layer dimensions compared to ordinary multiplication. We randomize the weight and input matrices and repeat the computation a hundred times for each architecture and average over the runs.

Practical Speedup



Practical Findings

- While there is a good speedup in computation for very wide networks (>5000 neurons), there is a slowdown for thinner networks. Thinner networks are more common in practice.
- Using two levels of recursion always performed worse than using just one level of recursion while taking much more memory.

Limitations

- Current implementations of Winograd’s Algorithm or Strassen’s Algorithm on the CPU as implemented in the BLAS-like Library Instantiation (BLIS) framework performed worse than optimized implementations in the Intel Math Kernel Library (MKL) on our tested architecture.

Future Work

- Using faster GPU implementations, such as (Huang et al. 2018, 2020).
- Using approximate matrix multiplication methods from randomized numerical linear algebra such as column-row sampling.
- Applying fast matrix multiplication with half-precision numbers used by a lot of deep learning architectures.

References

- J.-G. Dumas and V. Pan. Fast Matrix Multiplication and Symbolic Computation. 2016. arXiv preprint arXiv:cs.SC/1612.05766.
- J. Huang, C. D. Yu, and R. A. van de Geijn. Implementing Strassen's Algorithm with CUTLASS on NVIDIA Volta GPUs. 2018. arXiv preprint arXiv:cs.MS/1808.07984.
- P. Lai, H. Arafat, V. Elango, and P. Sadayappan. Accelerating Strassen-Winograd's matrix multiplication algorithm on GPUs. 2013. In the 20th Annual International Conference on High Performance Computing.
- J. Huang, C. D. Yu, and R. A. van de Geijn. Strassen's Algorithm Reloaded on GPUs. 2020. ACM Trans. Math. Softw. Article 1. ISSN 0098-3500.

I’d be glad to answer your questions and further discuss the paper on the conference platform or via email at akregeb@gmail.com.

Thank you!