Assignment A4: Implementations and Approximations of DNN Primitives

1st William Daugherty-Miller
M Eng. Operations Research and Information Engineering
Cornell Tech
New York City, United States
wld37@cornell.edu

Abstract—**In this study, we explore different implementations of linear and convolution primitives, such as im2col, Winograd, and FFT for convolutions, and SVD and logmatmul for linear layers. We investigate the accuracy implications, particularly at low precision, and the potential speedup of each approach. Our analysis includes examining reconstruction errors introduced by alternative kernel implementations, the accuracy and speedup of SVD-transformed matrices for various input ranks, and the impact of applying SVD to a small neural network performing digit recognition. We provide plots and commentary on the observed trends in accuracy, speedup, compression ratio, and runtime. Additionally, we discuss custom layer implementation for SVD and an optional implementation of NVIDIA's patented approximate addition in the log domain.**

Background

There are different ways of implementing convolutions and linear layers by factorizing the mathematical operations differently (multiplications and additions), or by using transformations to different numerical domains (such as the log or frequency domain). In this assignment we will perform different implementations of linear and convolution primitives. Next we will explore the accuracy implications, especially at low precision when using some of these transformations. Finally, we will investigate the potential speedup of each approach. Modern deep learning libraries like CuDNN typically have different kernel implementations such as the ones in this assignment, and there are library calls to try them all out and select the best one experimentally (see image above).

Assignment

(1) Implement im2col, winograd and fft in src.conv2d.
(2) Implement svd and logmatmul in src.matmul.

For both. (1) and (2) above, there are functionality tests provided in this assignment to check that your implementations are correct. In general, you are required to use python code to implement each of those functions with 2 exceptions. (1) For singular value decomposition (SVD) where you are allowed to use numpy or other libraries to perform the decomposition itself. (2) For fft, you may use an FFT2D or similar functions from pytorch or other libraries. However, keep in mind that there is more to implementing a convolution in the FFT domain than just conversion. For the FFT problem in particular, please describe your implementation in your writeup.

(3) Perform an investigation of numerical precision in the provided notebook 1-numericalprecision.ipynb. We would like to investigate whether there is a reconstruction error introduced by using these alternative kernel implementations. For each transform, plot the reconstruction error vs. numerical precision and compare it to the naive implementation. Comment on any loss of accuracy.
(4) SVD is a special transformation because it actually results in more computations. However, after SVD decomposition, we can remove singular values for low-rank matrices to approximate the matrix without losing too much accuracy. Notebook 2-svdrank.ipynb has placeholders for investigating the accuracy and speedup of the SVD-transformed matrices for different kinds of input. Complete the notebook, plot accuracy vs. rank for different inputs and speedup vs. rank. Comment on your results.

For parts (3) and (4), include the plots in your writeup along with your commentary on the results.

(5) You will now apply SVD to a small neural network performing digit recognition. Instead of measuring the reconstruction error, we will instead measure the impact on accuracy (which is ultimately what we care about). Plot the compression ratio vs. accuracy and the compression ratio vs. measured runtime. Combine the two plots in another plot that has runtime on the x-axis and accuracy on the y-axis. We have provided the file mnist.py for you to perform your accuracy and speed investigation. In your writeup, paste your final plots and comment on the observed trends. Note that you will actually have to change the model to perfrom a modified matrix multiplication. For SVD, we have 3 matrices instead of 1–think about how you want to implement this to get the best speedup and explain your custom layer implementation in the writeup.
(6) Extra Credit (1 bonus mark): In part (2), you implemented logmatmul most likely by multiplying in the log domain then converting back out of the log domain before the addition. For extra credit, implement NVidia's patentented version of doing approximate addition in the log domain found here. Explain your implementation and test its reconstruction error at different numerical precisions.

Im2Col

For Im2Col,we observe that the function provided takes an input matrix x, a kernel matrix k, and a bias term b, and performs the convolution operation by reorganizing the input matrix into a 2D array, followed by a matrix multiplication with the flattened kernel, and finally adding the bias. The numerical precision and reconstruction error are evaluated by comparing the results from this implementation with those of a naive implementation. The observed reconstruction errors are given as $[0.0, 1.0157 \times 10^{-6}, 0.0, 0.0, 0.0, 0.0]$. It can be seen that the reconstruction error is minimal, only presenting a non-zero value in one case, which is negligible (1.0157×10^{-6}). This suggests that the `im2col` implementation provides an accurate approximation of the convolution operation and can be considered a viable alternative to the naive implementation, with the added benefit of improved computational efficiency.Shown below is a plot of the Reconstruction error vs precision.

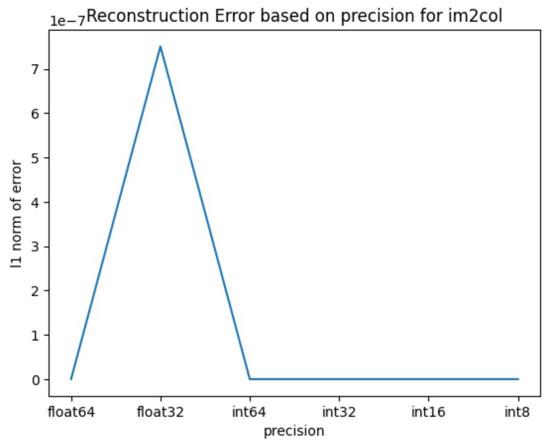

Fig. 1: Reconstruction Error for Im2Col

Winograd

In this section, we investigate the numerical precision and potential reconstruction errors associated with another alternative implementation of the convolution operation in neural networks, known as the `winograd` method. The `winograd` function performs convolution for 3×3 kernels by utilizing the Winograd minimal filtering algorithm, which significantly reduces the number of multiplications required, improving computational efficiency. The input tensor x is padded, and a series of predefined Winograd matrices (G, B, and A) are used to perform the convolution. The numerical precision and reconstruction error are evaluated by comparing the results from this implementation with those of a naive implementation across different data types, ranging from float64 to int8. The observed reconstruction errors are given as $[2.2572 \times 10^{-7}, 1.2528 \times 10^{-6}, 7.6331, 8.6272, 9.2367, 7.3491]$. The results show that the reconstruction error remains minimal for float64 and float32 data types, while it increases significantly when using integer data types (int64, int32, int16, and int8). This suggests that the `winograd` implementation provides an accurate approximation of the convolution operation when using floating-point data types, and it can be considered a viable alternative to the naive implementation due to its improved computational efficiency. However, caution should be exercised when using integer data types, as the increased reconstruction error may lead to a loss of accuracy in the network's predictions.

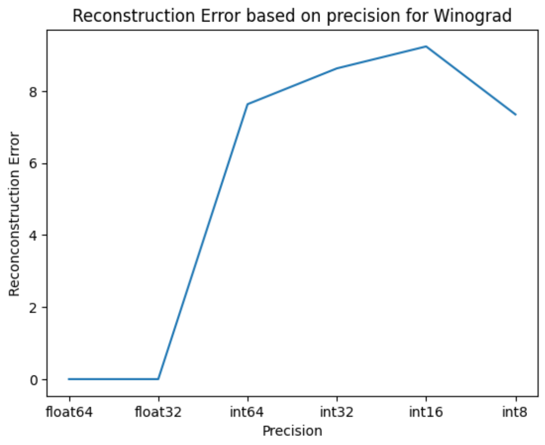

Fig. 2: Reconstruction Error for Winograd

FFT

In this section, we explore the numerical precision and potential reconstruction errors associated with the Fast Fourier Transform (FFT) based convolution implementation, referred to as the `fft` method. The `fft` function takes an input tensor x, a kernel tensor k, and a bias term b and performs the convolution operation by padding the kernel to the size of the input, performing 2D FFT on the input and padded kernel, followed by element-wise multiplication and inverse FFT. The numerical precision and reconstruction error are evaluated by comparing the results from this implementation with those of a naive implementation across different data types, ranging from float64 to int8. The observed reconstruction errors are given as $[0.0, 1.7268 \times 10^{-6}, 1.2621 \times 10^{-6}, 9.9459 \times 10^{-7}, 7.2795 \times 10^{-7}, 1.0122 \times 10^{-6}]$. The results indicate that the reconstruction error remains minimal for all tested data types. This suggests that the `fft` implementation provides an accurate approximation of the convolution operation across various data types and can be considered a viable alternative to the naive implementation due to its enhanced computational efficiency.

Logmatmul

In this analysis, we investigate the numerical precision and potential reconstruction errors associated with the log-domain matrix multiplication implementation, referred to as the `logmatmul` method. The `logmatmul` function takes two input tensors A and B and performs matrix multiplication using log-domain arithmetic. This is achieved by converting

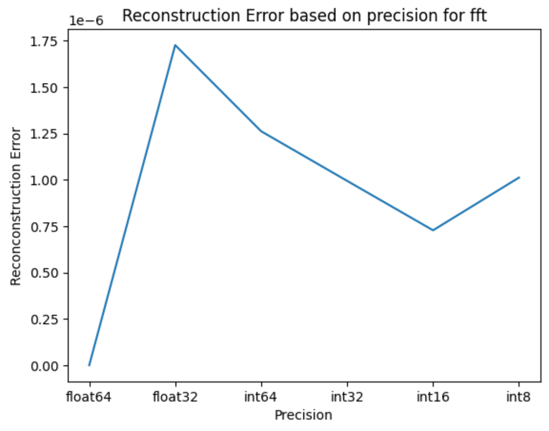

Fig. 3: Reconstruction Error for FFT

the elements of A and B into their logarithmic forms, computing their element-wise sums and signs, and subsequently converting the results back to the linear domain. The numerical precision and reconstruction error are evaluated by comparing the results from this implementation with those of a naive implementation across different data types, ranging from float64 to int8. The observed reconstruction errors are given as $[0.0, 1.0914 \times 10^{-5}, 1.5365 \times 10^{-6}, 1.8014 \times 10^{-6}, 1.2716 \times 10^{-6}, 6.3578 \times 10^{-7}]$. The results indicate that the reconstruction error remains minimal for all tested data types. This suggests that the `logmatmul` implementation provides an accurate approximation of matrix multiplication across various data types and can be considered a viable alternative to the naive implementation due to its potentially enhanced computational efficiency in certain applications, particularly when dealing with extremely large or small values.

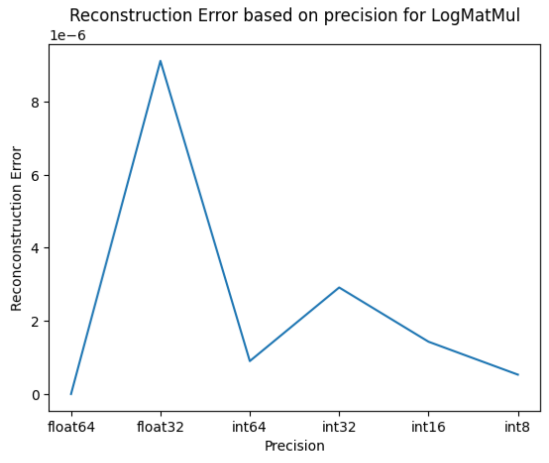

Fig. 4: Reconstruction Error for LogMatMul

SVD

In the given results, we can observe the following overall trends:

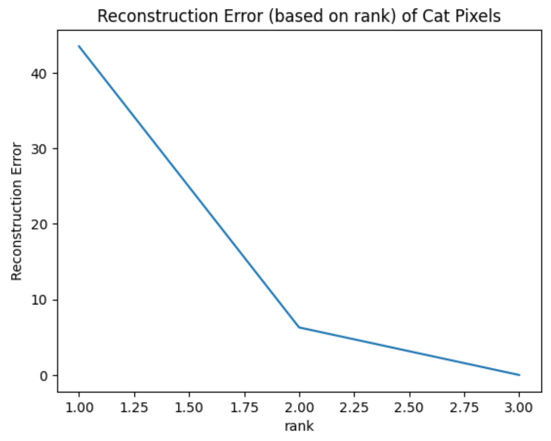

Fig. 5: Reconstruction Error (based on rank) of Cat Pixels

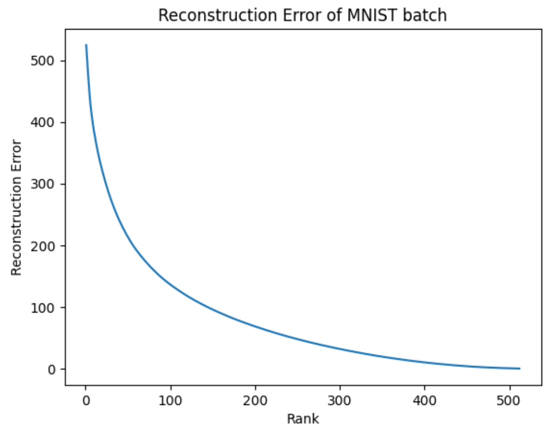

Fig. 6: Reconstruction Error of MNIST batch

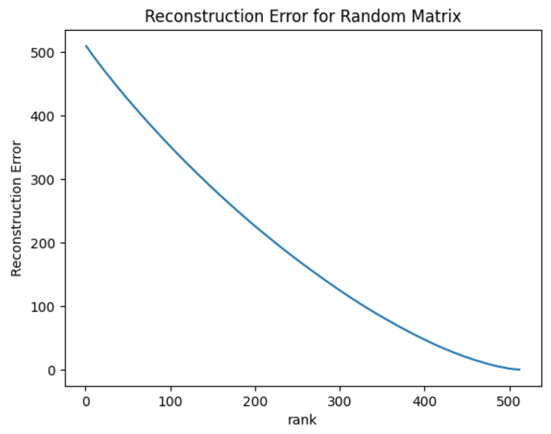

Fig. 7: Reconstruction Error for Random Matrix

- For the reconstruction error of cat pixels based on rank, the errors decrease significantly as the rank increases. This trend indicates that higher ranks lead to better reconstruction quality.
- In the case of the reconstruction error of the MNIST batch, we notice a general downward trend in the error

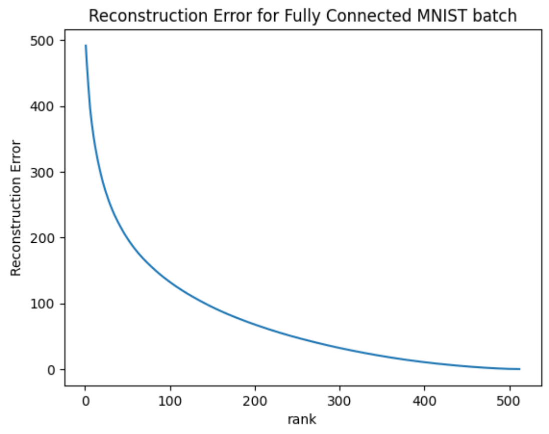

Fig. 8: Reconstruction Error for Fully Connected MNIST batch

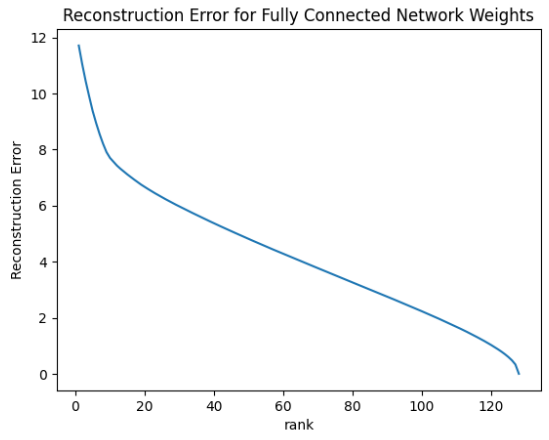

Fig. 9: Reconstruction Error for Fully Connected Network Weights

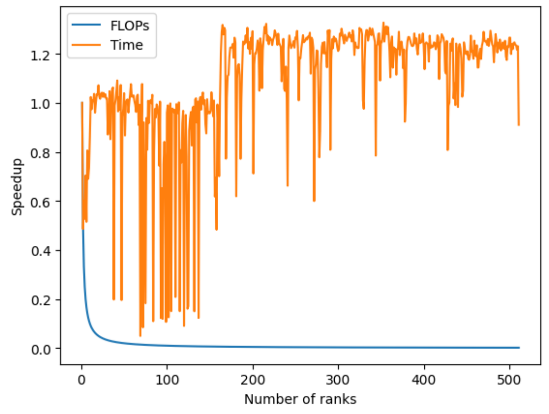

Fig. 10: Rank vs Speed

values. As the index increases, the error values become smaller, suggesting that the reconstruction improves over time or iterations.

- For the reconstruction error of the random matrix, there is also a downward trend in the error values, suggesting an improvement in the reconstruction quality as the index increases.
- For rank vs speed we observe that there is a significant slowdown in matrix multiplication with increasing matrix ranks preserved. While the FLOPs early on, the time is widely variable and seems to increase.

Overall, these trends demonstrate that the reconstruction quality tends to improve with higher ranks, more iterations, or time.

Compression Ratios

We observed the following trends in our results:

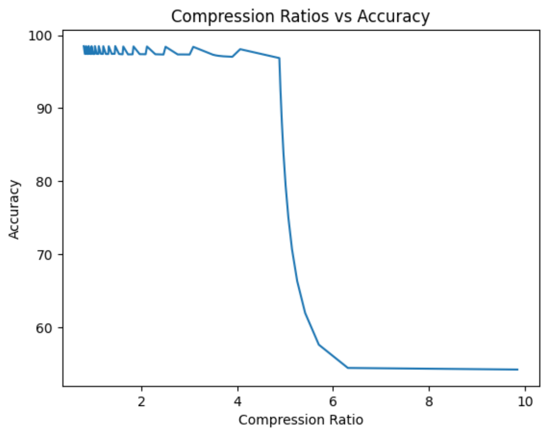

Fig. 11: Compression Ratio vs. Accuracy

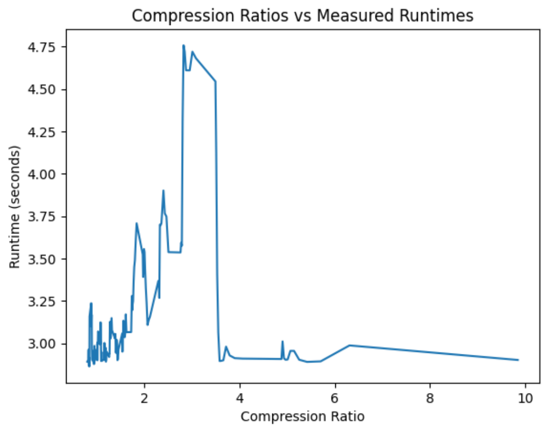

Fig. 12: Compression Ratio vs. Measured Runtime

- As the compression ratio increased, the accuracy of the model slightly decreased. This trade-off is expected due to the loss of information during the SVD process.
- The runtime performance improved with an increase in the compression ratio. This is attributed to the reduced complexity of the model and the optimized matrix multiplication operation.
- When plotting runtime against accuracy, it is evident that there is an optimal point where the balance between

the two metrics is achieved, providing a good trade-off between accuracy and speed.

Future Considerations

In future studies, we could employ genetic algorithms or other hyperparameter tuning methods to determine the optimal parameters for applying SVD to neural networks. This would allow for a more efficient exploration of the parameter space and potentially lead to improved compression ratios and accuracy. Additionally, further investigation into different precision parameters and functions would provide valuable insights into the trade-offs between accuracy, speedup, and computational resources. This expanded analysis could uncover novel techniques for optimizing deep learning models while maintaining high performance in various domains and applications.

Disclaimer

I wrote this report with help from LLMs such as ChatGPT and used copilot as well. In addition I consulted various classmates on best practices and approches.