Optimizing Linear/Fully-Connected Layers

User's Guide | NVIDIA Docs

DU-09800-001_v001 | February 2023

Table of Contents

List of Figures

List of Tables

Chapter 1. Quick Start Checklist

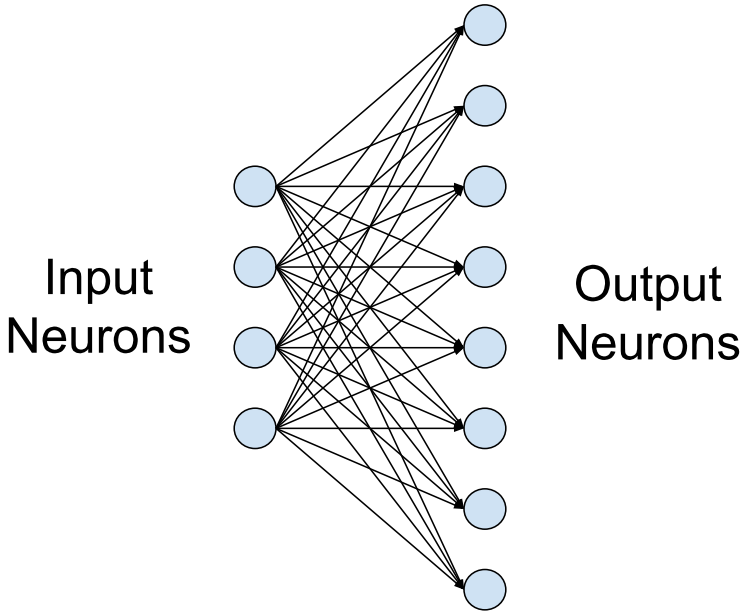
The following quick start checklist provides specific tips for fully-connected layers.

- Choose the batch size and the number of inputs and outputs to be divisible by 4 (TF32) / 8 (FP16) / 16 (INT8) to run efficiently on Tensor Cores. For best efficiency on A100, choose these parameters to be divisible by 32 (TF32) / 64 (FP16) / 128 (INT8) refer to Tensor Core Requirements.
- Especially when one or more parameters are small, choosing the batch size and the number of inputs and outputs to be divisible by at least 64 and ideally 256 can streamline tiling and reduce overhead; refer to Dimension Quantization Effects.
- Larger values for batch size and the number of inputs and outputs improve parallelization and efficiency; see Performance and its subsections.
- As a rough guideline, choose batch sizes and neuron counts greater than 128 to avoid being limited by memory bandwidth (NVIDIA® A100-SXM4-80GB; this threshold is similar for other A100 and V100 GPUs); see Batch Size.

Chapter 2. Fully-Connected Layer

Fully-connected layers, also known as linear layers, connect every input neuron to every output neuron and are commonly used in neural networks.

Figure 1. Example of a small fully-connected layer with four input and eight output neurons.



Three parameters define a fully-connected layer: batch size, number of inputs, and number of outputs. Forward propagation, activation gradient computation, and weight gradient computation are directly expressed as matrix-matrix multiplications. How the three parameters map to GEMM dimensions (General Matrix Multiplication, background in the NVIDIA Matrix Multiplication Background User's Guide) varies among frameworks, but the underlying principles are the same. For the purposes of the discussion, we adopt the

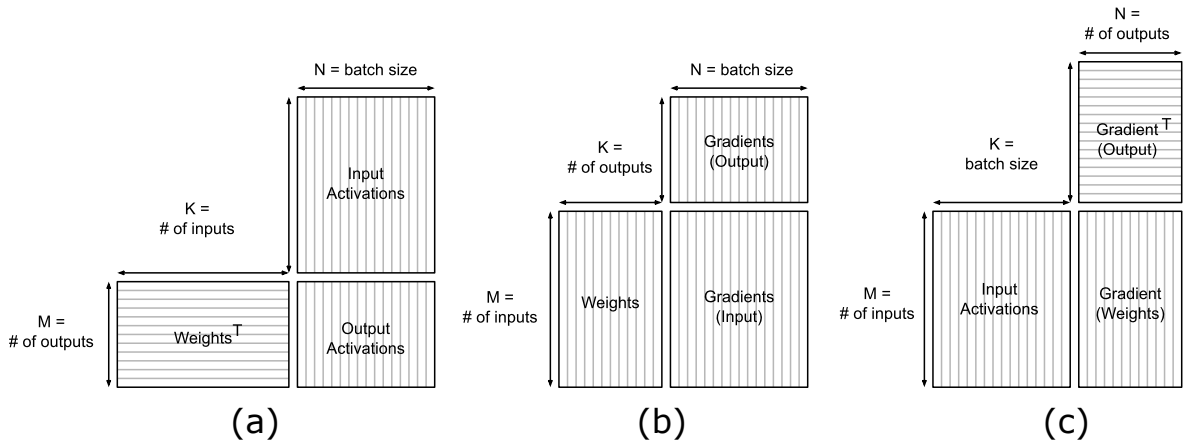
convention used by PyTorch and Caffe where A contains the weights and B the activations. In TensorFlow, matrices take the opposite roles, but the performance principles are the same.

Table 1. Mapping of inputs, outputs, and batch size to GEMM parameters M, N, K.

Computation Phase	M	N	K
Forward Propagation	Number of outputs	Batch size	Number of inputs
Activation Gradient	Number of inputs	Batch size	Number of outputs
Weight Gradient	Number of inputs	Number of outputs	Batch size

The compositions of the matrices in the GEMM are shown in Figure 2.

Figure 2. Dimensions of equivalent GEMMs for (a) forward propagation, (b) activation gradient, and (c) weight gradient computations of a fully-connected layer.

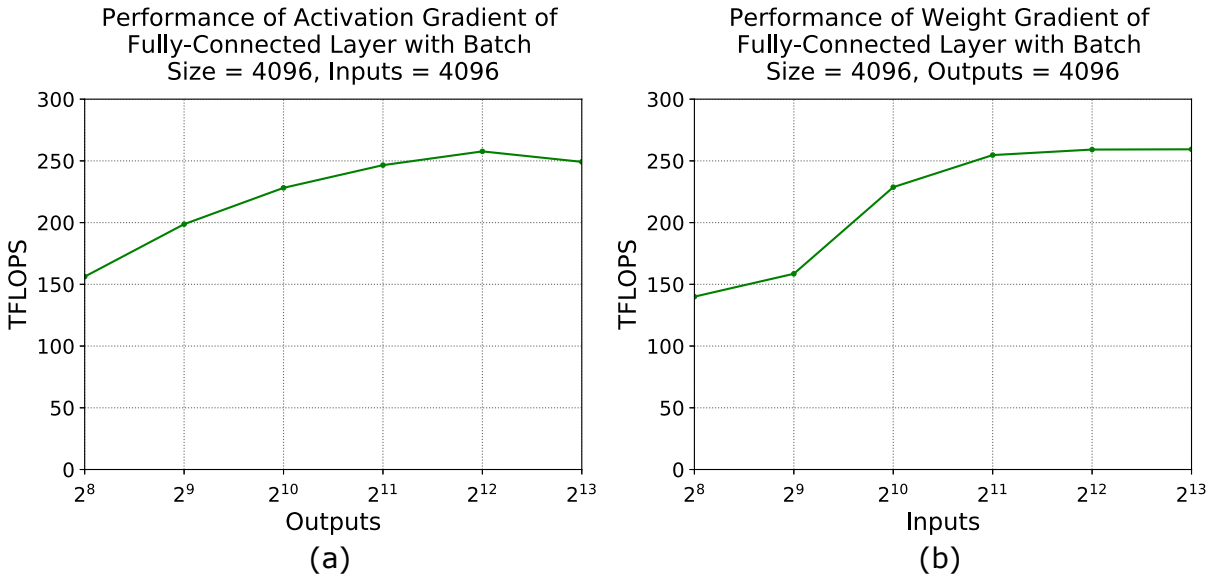


Chapter 3. Performance

3.1. Input Features And Output Neuron Counts

As fully-connected layers directly correspond to GEMMs, their performance trends are identical to those described in Typical Tile Dimensions In NVIDIA cuBLAS And Performance. Larger parameters tend to allow better parallelization and efficiency; a GEMM that is twice the size often takes less than twice the time to calculate.

Figure 3. Larger fully-connected layers are equivalent to larger GEMMs, which perform better. NVIDIA A100-SXM4-80GB, CUDA 11.2, cuBLAS 11.4.

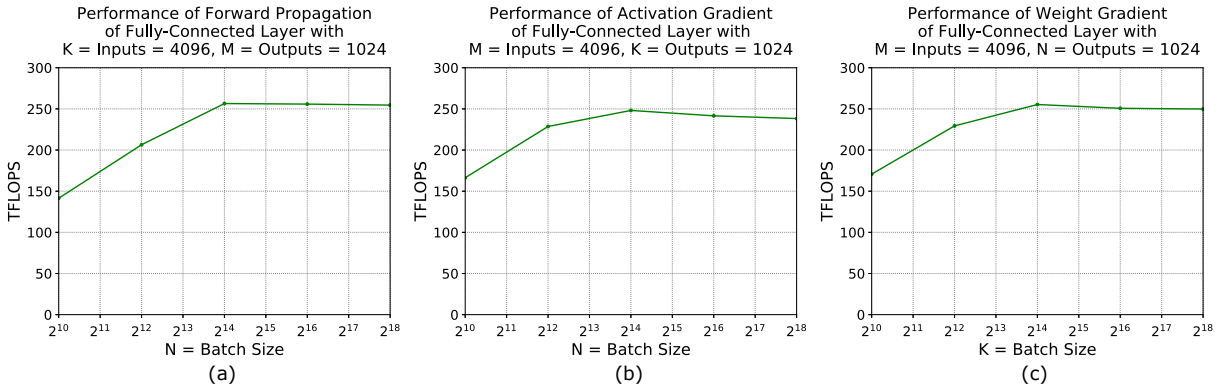


3.2. Batch Size

The batch size directly contributes to the tiling strategy for two out of three training phases - forward pass and activation gradient computation. For these phases, the output matrix dimension includes batch size, so larger batch sizes result in more tiles. Training with larger batch sizes is one option to extract more performance when model size is too small to fully utilize a GPU.

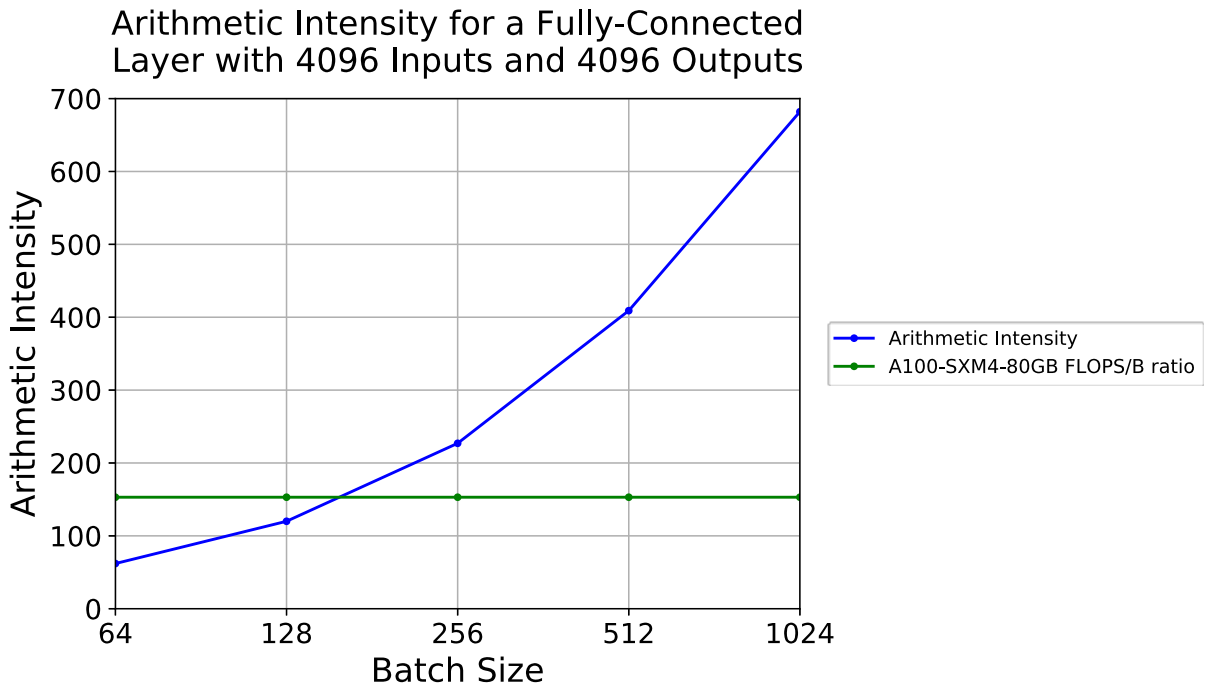
For weight gradient computation, the output matrix has the same dimensions as the weights, thus batch size does not affect the tile count directly. Instead, batch size here maps to the K dimension of the GEMM; larger batch size enables more efficient computation per tile of weight gradients. Figure 15 shows the performance impact of varying batch size on forward, activation gradient, and weight gradient computations for a fully-connected layer with 4096 inputs and 1024 outputs. The larger batch sizes yield roughly 250 TFLOPS delivered performance.

Figure 4. Performance data for (a) forward propagation, (b) activation gradient computation, and (c) weight gradient computation for a fully-connected layer with 4096 inputs, 1024 outputs, and varying batch size. NVIDIA A100-SXM4-80GB, CUDA 11.2, cuBLAS 11.4.



Of particular interest are GEMMs where one dimension is very small. For example, on NVIDIA A100-SXM4-80GB and for a fully-connected layer with 4096 inputs and 4096 outputs, forward propagation, activation gradient computation, and weight gradient computation are estimated to be memory-bound for batch sizes 128 and below (see Figure 5).

Figure 5. Arithmetic intensity for a fully-connected layer with 4096 inputs and 4096 outputs. Batch sizes 128 and below are bandwidth limited on NVIDIA A100 accelerators.



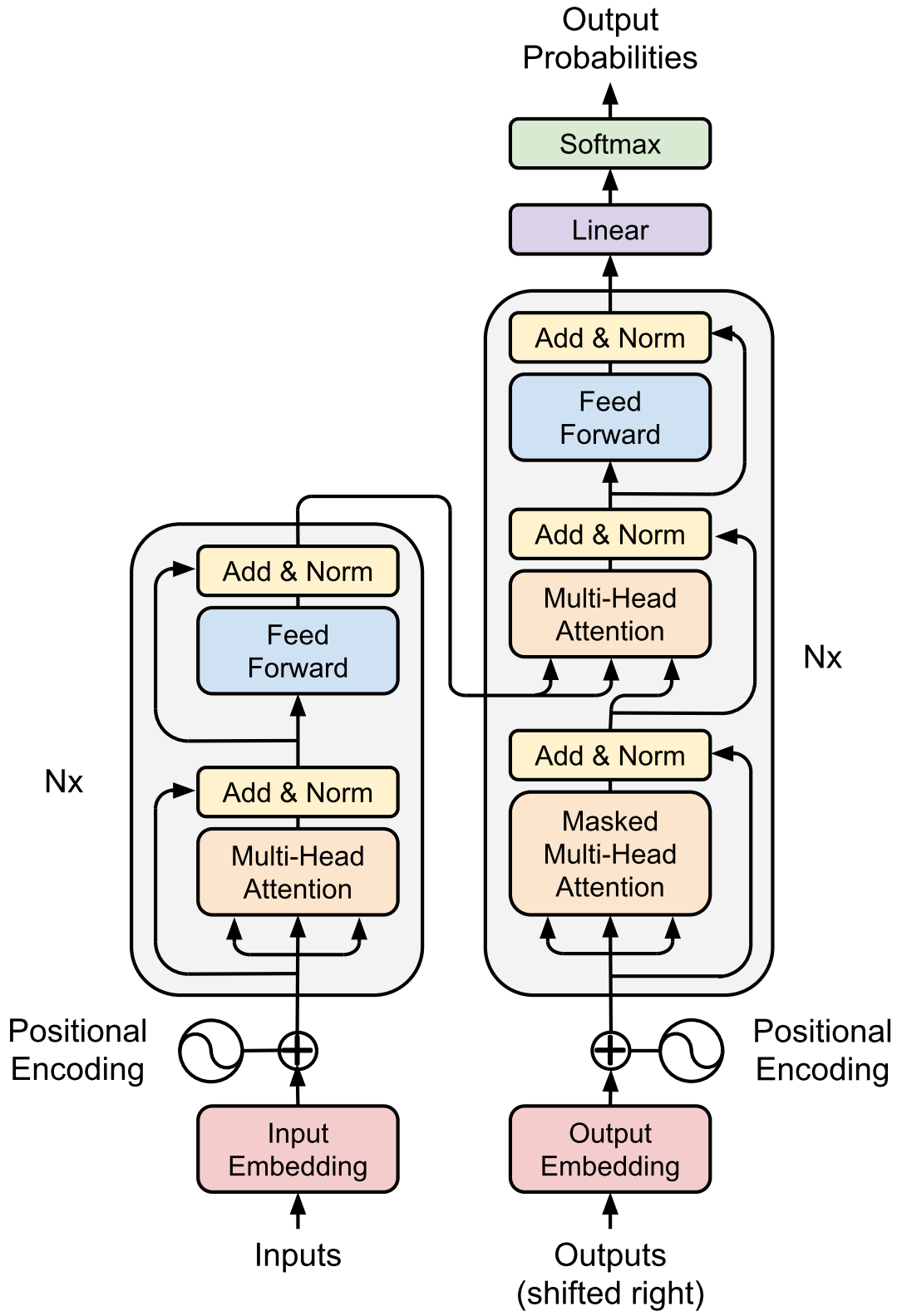
Larger numbers of inputs and outputs improve performance somewhat, but the computation will always be bandwidth-limited for very small batch sizes, for example, 8 and below. For a discussion of math- and bandwidth-limited computations, refer to Math And Memory Bounds.

Chapter 4. Transformer Case Study

4.1. Basics

Transformers are a popular neural network architecture used for sequence-to-sequence mapping tasks, for example for natural language translation. They use an encoder-decoder architecture making heavy use of attention, both to "self-attend" over input sequences, as well as to give the decoder access to the encoder's context. Figure 6 shows the complete neural network architecture (Attention Is All You Need 2017 paper, page 3).

Figure 6. Transformer neural network architecture with N macro-layers in the encoder and decoder, respectively. Macro-layers consist of an attention layer(s) and a feed-forward network.



From a performance standpoint, Transformers fundamentally process all the tokens in an input sequence in parallel, unlike - for example - RNN architectures with their sequential dependency. That makes Transformers very amenable to highly parallel architectures such as GPUs, and leads to large GEMMs that, with a few simple guidelines, can take great advantage of Tensor Core acceleration.

4.2. Applying Tensor Core Guidelines

4.2.1. Step 1: Padding The Vocabulary Size

Consider the final linear layer in the Transformer network, whose number of outputs is equal to the vocabulary size, as it is feeding the final SoftMax layer in the network to produce a probability distribution across tokens in the vocabulary.

This linear layer, as discussed in the Optimizing Fully-Connected Layers User's Guide, has M equal to the vocabulary size, N equal to the batch size, and K equal to the input feature size (all in the forward pass). Because the vocabulary is usually large, this is a heavyweight computation, and it is important to ensure Tensor Cores are being used effectively.

Figure 7. Performance benefits substantially from choosing vocabulary size to be a multiple of 8 with both (a) cuBLAS version 10.1 and (b) cuBLAS version 11.0. The projection layer uses 1024 inputs and a batch size of 5120. NVIDIA V100-SXM2-16GB GPU.

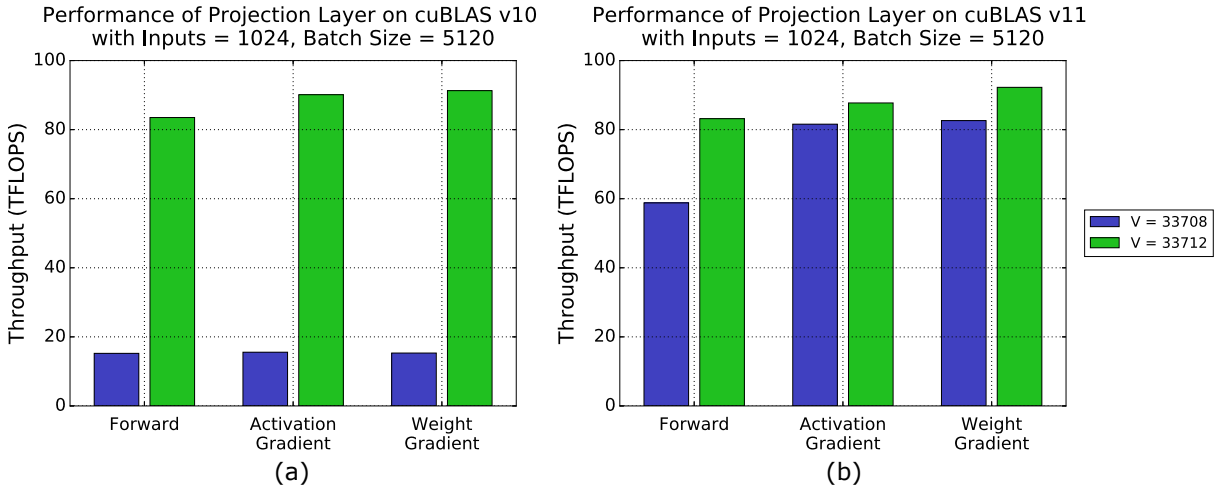


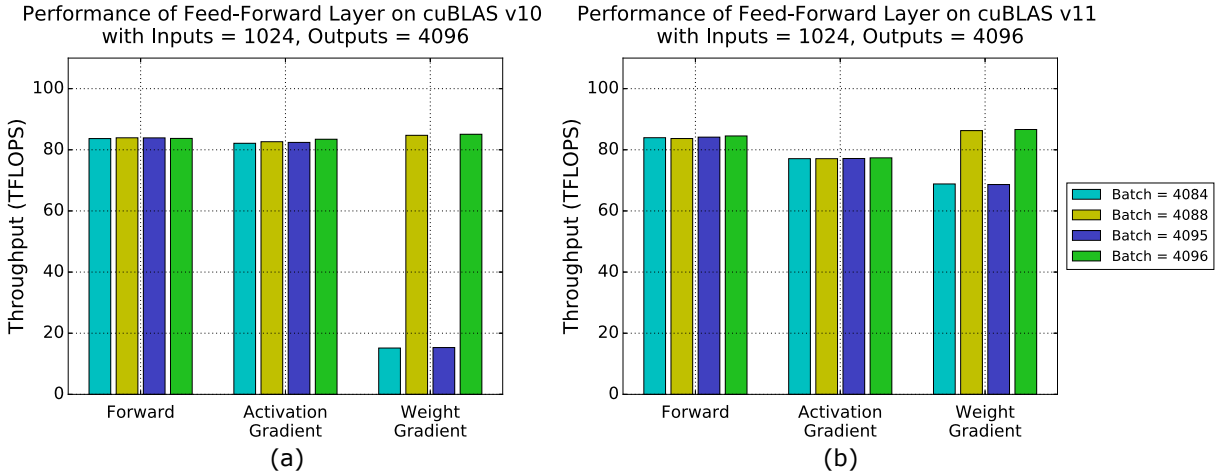
Figure 7 shows what happens when the vocabulary size is chosen without regard to alignment. FP16 data is used, so dimensions must be multiples of 8 for best alignment. This is most important when using a cuBLAS version lower than 11.0 (Figure 7 (a)); in this case, when vocabulary size is not divisible by 8 (V=33708), Tensor Cores cannot be applied and performance reduces drastically to the levels sustained by the NVIDIA CUDA® cores. Simply adding four padding tokens (to reach V=33712) switches to a multiple-of-8 size and

dramatically accelerates the overall computation. When using cuBLAS 11.0 or higher (Figure 7 (b)), performance impact is not as extreme, but choosing vocabulary size to be aligned to a multiple of 8 is still noticeably more efficient. For more detail on alignment and efficiency, refer to Tensor Core Requirements.

4.2.2. Step 2: Choosing Multiple-Of-8 Batch Sizes

Besides the projection layer near the end of the network, fully-connected layers are a major Transformer building block in all other parts of the network as well, including the big self-attention and feed-forward blocks. As described before, batch size directly maps to one of the GEMM dimensions in such layers - N in the forward and activation gradient passes, K in the weight gradient pass - and therefore, the guideline to pad to a multiple of 8 applies to batch size as well.

Figure 8. Weight gradient calculation for a fully-connected layer benefits from padding batch size to be a multiple of 8 with both (a) cuBLAS version 10.1 and (b) cuBLAS version 11.0. The first fully-connected layer (4096 outputs, 1024 inputs) from the Transformer feed-forward network is shown. NVIDIA V100-SXM2-16GB GPU.



The effect from padding batch size on one of the fully-connected layers in the network is shown in Figure 8. Here, we've picked the first layer in the feed-forward block, which is a fully-connected layer with 1024 inputs and 4096 outputs. As the chart shows, this is an example where the multiple-of-8 rule does not necessarily need to be applied to all three GEMM dimensions; both forward and activation gradient passes perform the same with and without padding. The weight gradient pass, on the other hand, shows the same performance difference we saw on the projection GEMM earlier. As in that example, for cuBLAS versions lower than 11.0 (Figure 8 (a)), performance improvement is dramatic: with a batch size of 4095 tokens, CUDA cores are used as a fallback, whereas a batch size of 4096 tokens enables Tensor Core acceleration. When using cuBLAS 11.0 and higher (Figure 8 (b)), performance improvement is

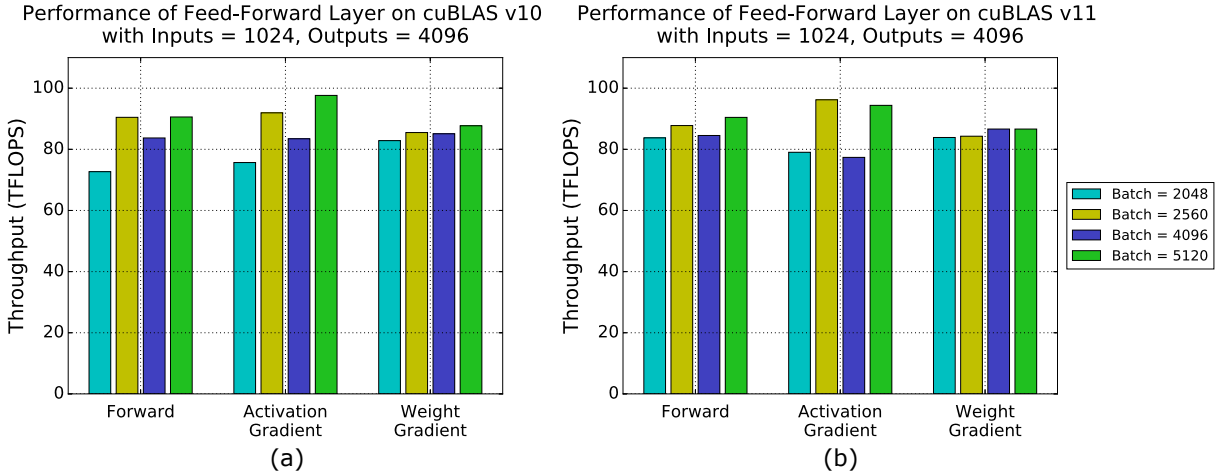
less extreme, but still significant. We recommend ensuring all three GEMM dimensions are multiples of 8 when training in FP16 so that all passes will use Tensor Cores efficiently.

4.2.3. Step 3: Avoiding Wave Quantization Through Batch Size Choice

Because batch size directly controls the shape of the MxN output matrix and Tensor Core GEMMs are parallelized by tiling the output matrix, choosing batch size appropriately can be used to reduce tile and wave quantization effects.

For the Transformer, let us consider the first layer in the feed-forward block again (4096 outputs, 1024 inputs). In this layer, the output matrix is of shape 4096 x batch size. Assuming a tile size of 256x128 as an example, the `M=4096` dimension results in `4096/256=16` thread block tiles stacked vertically. On an NVIDIA V100 GPU with 80 SMs, wave quantization is minimal if the total number of thread blocks is a multiple of 80 (or just below). Therefore, choosing the batch size to result in `n*80/16=n*5` thread block tiles in the N dimension achieves optimal wave quantization. With 256x128 thread blocks, this is achieved by choosing batch sizes of `N=1*5*128=640`, `N=2*5*128=1280`, and so on. Figure 9 illustrates the effect this has using two common batch sizes, 2048 and 4096.

Figure 9. Fully-connected layer performance benefits from eliminating wave quantization by choosing batch size appropriately; improvement is similar with (a) cuBLAS version 10.1 and (b) cuBLAS version 11.0. The first fully-connected layer from the feed-forward block is shown as an example. Batch sizes 2560 and 5120 result in a multiple of 80 thread blocks running on an 80-SM NVIDIA V100 GPU. In the weight gradient, batch size maps to the K parameter of the GEMM and hence does not control the shape of the output matrix or have any immediate effect on wave quantization. NVIDIA V100-SXM2-16GB GPU.



The chart shows that choosing a quantization-free batch size (2560 instead of 2048, 5120 instead of 4096) can noticeably improve performance. In particular, it is noteworthy that batch size 2560 (resulting in 4 waves of 80 thread block tiles each, assuming 256x128 tile size) *achieves higher throughput than the larger batch size of 4096* (512 thread block tiles, 6.4 waves with 256x128 tile size). The activation gradient with batch size 5120 achieves about 95 TFLOPS delivered performance. For the weight gradient computation, batch size maps to the K parameter of the GEMM, and hence does not directly influence the size and shape of the output matrix or the number of thread block tiles that are created.

Notice

This document is provided for information purposes only and shall not be regarded as a warranty of a certain functionality, condition, or quality of a product. NVIDIA Corporation ("NVIDIA") makes no representations or warranties, expressed or implied, as to the accuracy or completeness of the information contained in this document and assumes no responsibility for any errors contained herein. NVIDIA shall have no liability for the consequences or use of such information or for any infringement of patents or other rights of third parties that may result from its use. This document is not a commitment to develop, release, or deliver any Material (defined below), code, or functionality.

NVIDIA reserves the right to make corrections, modifications, enhancements, improvements, and any other changes to this document, at any time without notice.

Customer should obtain the latest relevant information before placing orders and should verify that such information is current and complete.

NVIDIA products are sold subject to the NVIDIA standard terms and conditions of sale supplied at the time of order acknowledgement, unless otherwise agreed in an individual sales agreement signed by authorized representatives of NVIDIA and customer ("Terms of Sale"). NVIDIA hereby expressly objects to applying any customer general terms and conditions with regards to the purchase of the NVIDIA product referenced in this document. No contractual obligations are formed either directly or indirectly by this document.

NVIDIA products are not designed, authorized, or warranted to be suitable for use in medical, military, aircraft, space, or life support equipment, nor in applications where failure or malfunction of the NVIDIA product can reasonably be expected to result in personal injury, death, or property or environmental damage. NVIDIA accepts no liability for inclusion and/or use of NVIDIA products in such equipment or applications and therefore such inclusion and/or use is at customer's own risk.

NVIDIA makes no representation or warranty that products based on this document will be suitable for any specified use. Testing of all parameters of each product is not necessarily performed by NVIDIA. It is customer's sole responsibility to evaluate and determine the applicability of any information contained in this document, ensure the product is suitable and fit for the application planned by customer, and perform the necessary testing for the application in order to avoid a default of the application or the product. Weaknesses in customer's product designs may affect the quality and reliability of the NVIDIA product and may result in additional or different conditions and/or requirements beyond those contained in this document. NVIDIA accepts no liability related to any default, damage, costs, or problem which may be based on or attributable to: (i) the use of the NVIDIA product in any manner that is contrary to this document or (ii) customer product designs.

No license, either expressed or implied, is granted under any NVIDIA patent right, copyright, or other NVIDIA intellectual property right under this document. Information published by NVIDIA regarding third-party products or services does not constitute a license from NVIDIA to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property rights of the third party, or a license from NVIDIA under the patents or other intellectual property rights of NVIDIA.

Reproduction of information in this document is permissible only if approved in advance by NVIDIA in writing, reproduced without alteration and in full compliance with all applicable export laws and regulations, and accompanied by all associated conditions, limitations, and notices.

THIS DOCUMENT AND ALL NVIDIA DESIGN SPECIFICATIONS, REFERENCE BOARDS, FILES, DRAWINGS, DIAGNOSTICS, LISTS, AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, "MATERIALS") ARE BEING PROVIDED "AS IS." NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE MATERIALS, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE. TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL NVIDIA BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF NVIDIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Notwithstanding any damages that customer might incur for any reason whatsoever, NVIDIA's aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms of Sale for the product.

Google

Android, Android TV, Google Play and the Google Play logo are trademarks of Google, Inc.

NVIDIA Corporation | 2788 San Tomas Expressway, Santa Clara, CA 95051
www.nvidia.com

Trademarks

NVIDIA Corporation | 2788 San Tomas Expressway, Santa Clara, CA 95051
www.nvidia.com