Optimizing Memory-Limited Layers

User's Guide | NVIDIA Docs

DU-09797-001_v001 | February 2023

Table of Contents

List of Figures

Chapter 1. Quick Start Checklist

The following quick start checklist provides specific tips for layers whose performance is limited by memory accesses.

- Explore the available implementations of each layer in the NVIDIA cuDNN API Reference or your framework. Often the best way to improve performance is to choose a more efficient implementation. For example, persistent implementations of batch normalization require fewer loads from memory.
- Be aware of the number of memory accesses required for each layer. Performance of a memory-bound calculation is simply based on the number of inputs, outputs, and weights that need to be loaded and/or stored per pass. We don't have recommended parameter tweaks for these layers.
- Be aware of the impact of each layer on the overall training step performance. Memory-bound layers are most likely to take a significant amount of time in small networks where there are no large and computation-heavy layers to dominate performance.

Chapter 2. Memory-Limited Layers

Many types of layers used in deep learning models, including normalization, activation functions, and pooling layers, involve relatively few calculations per input and output value. On the GPU, forward and backward propagation of these layers is expected to be limited by memory transfer times.

The reasons behind this are explained in greater detail in the NVIDIA GPU Performance Background User's Guide. This guide focuses on performance trends common among memory-limited layers and any important algorithm and parameter choices for each.

Chapter 3. Normalization

Normalization layers are a popular tool to improve regularization in training.

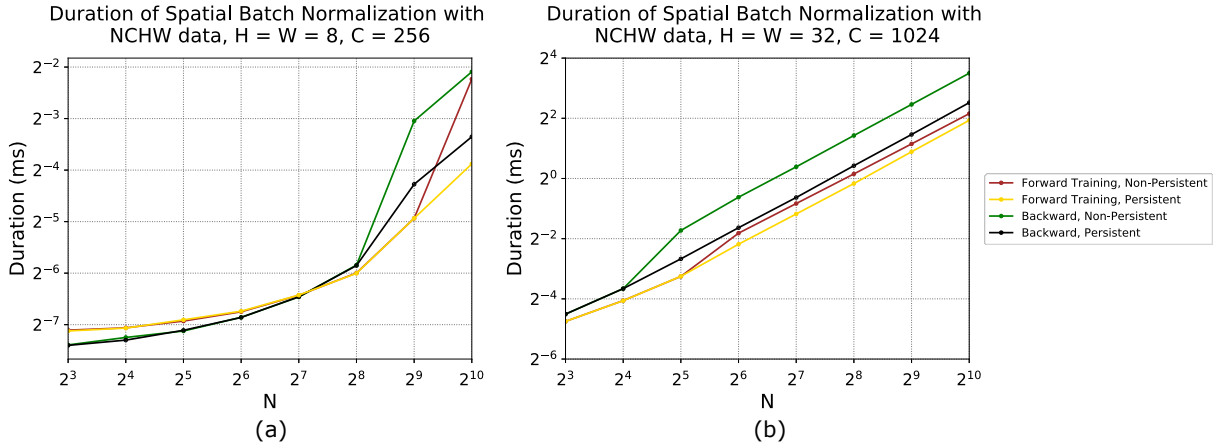
There are many variants of normalization operations, differing in the "region" of the input tensor that is being operated on (for example, batch normalization operating on all pixels within a color channel, and layer normalization operating on all pixels within a mini-batch sample). All these operations have very similar performance behavior; they are all limited by memory bandwidth. As an example, let's consider batch normalization.

3.1. Batch Normalization

Batch normalization (BN) layers take a 4D (NCHW or other layout) tensor as input, normalize, scale, and shift all the pixels within each channel C. In most convolutional neural networks, BN layers follow after a convolutional layer.

Batch normalization does not have enough operations per value in the input tensor to be math limited on any modern GPU; the time taken to perform the batch normalization is therefore primarily determined by the size of the input tensor and the available memory bandwidth.

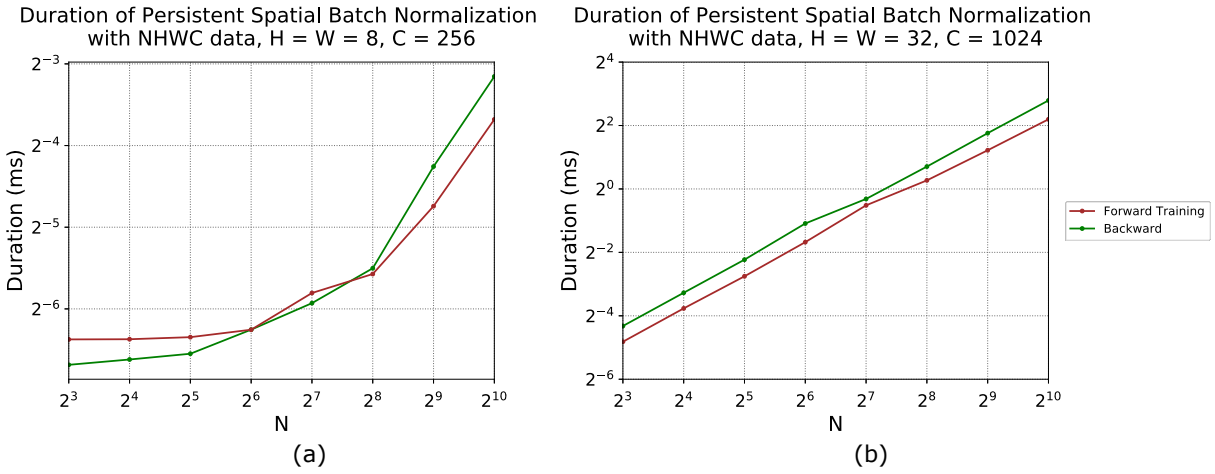
Figure 1. Duration of spatial and persistent spatial batch normalization in two different regions with NCHW data. Note that both axes are logarithmically scaled. NVIDIA A100-SXM4-80GB, CUDA 11.2, cuDNN 8.1.



When input tensors are very small, duration does not change with input size (Figure 1 (a), with batch size below 64 for forward training). This is due to tensors being small enough that memory bandwidth isn't fully utilized. However, for larger inputs, the duration increases close to linearly with size (Figure 1 (b)); it will take twice as long to move twice as many input and output values.

Two different algorithm options are benchmarked Figure 1. Non-persistent batch normalization is a multi-pass algorithm, where input data will be read one or more times to compute statistics such as mean and variance, then read again to be normalized. When inputs are small enough, a better single-pass algorithm (persistent batch normalization) can be used by cuDNN - here, inputs are read once into on-chip GPU memory, and then both statistics computation and normalization is performed from there, without any additional data reads. Fewer data reads result in reduced traffic to off-chip memory, which - for constant bandwidth - means the duration is reduced. In other words, spatial persistent batch normalization is faster than its non-persistent variant.

Figure 2. Duration of persistent spatial batch normalization in two different regions, this time with NHWC data. Note that both axes are logarithmically scaled. NVIDIA A100-SXM4-80GB, CUDA 11.2, cuDNN 8.1.



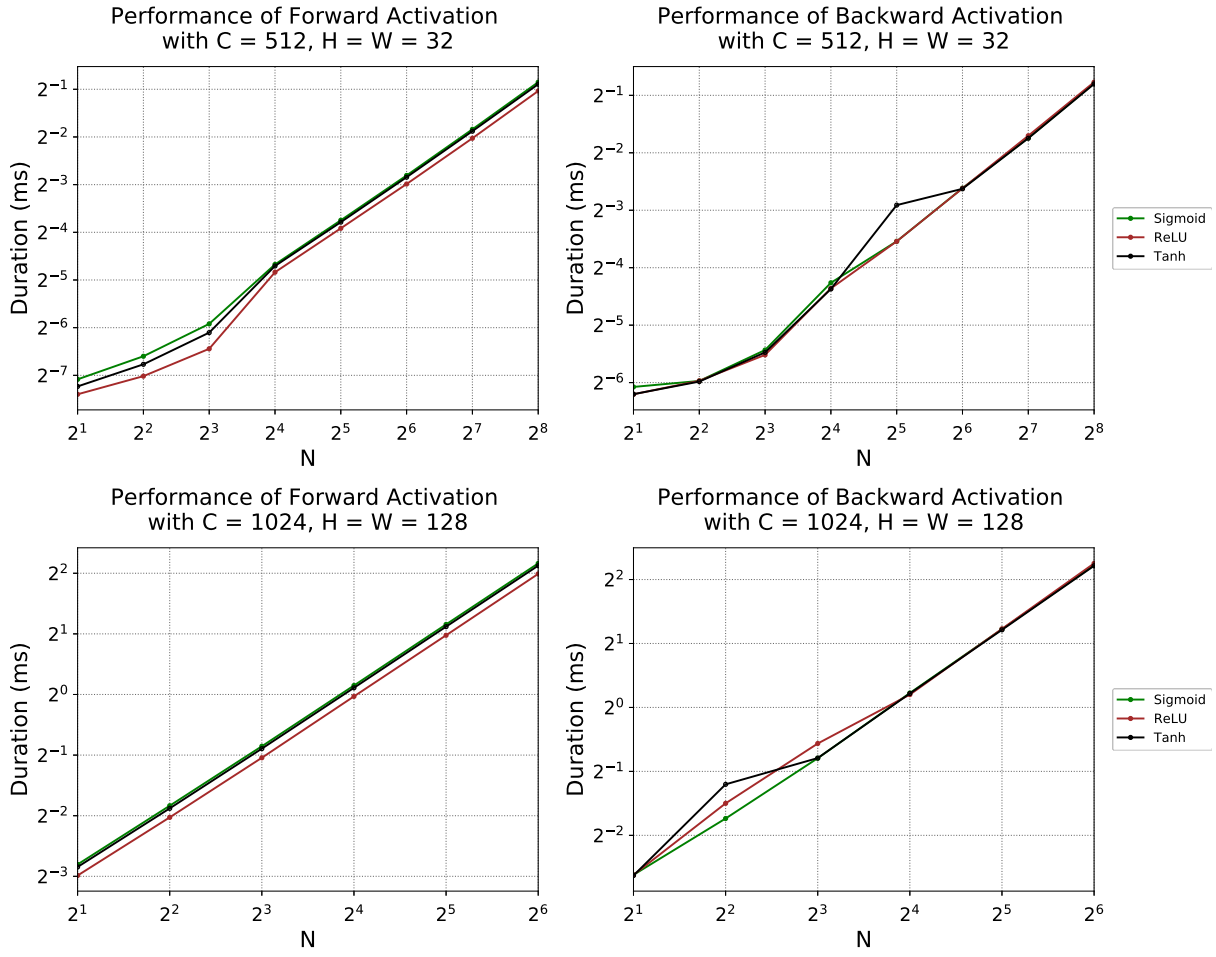
Similar trends can be seen when NHWC data is used.

Chapter 4. Activations

Activation functions typically follow a fully-connected, convolutional, or recurrent layer in a network. These functions are applied to each activation value independently, hence we refer to them as "element-wise" operations.

The shape of the activation tensor remains unchanged. For details on a possible implementation of activation functions, see the documentation for cudnnActivationForward() and cudnnActivationBackward(). Deep learning frameworks such as TensorFlow and PyTorch often use their own (non-cuDNN) implementations and libraries to execute activation functions, for example via the Eigen library in TensorFlow 1.13. Regardless of implementation, the general performance behavior of activation functions (and other element-wise operations) on the GPU is always the same, as we'll describe below.

Figure 3. Duration of forward and backward propagation of activation functions is proportional to input size (N*H*W*C here). Note that both axes are scaled logarithmically. NVIDIA A100-SXM4-80GB, CUDA 11.2, cuDNN 8.1.

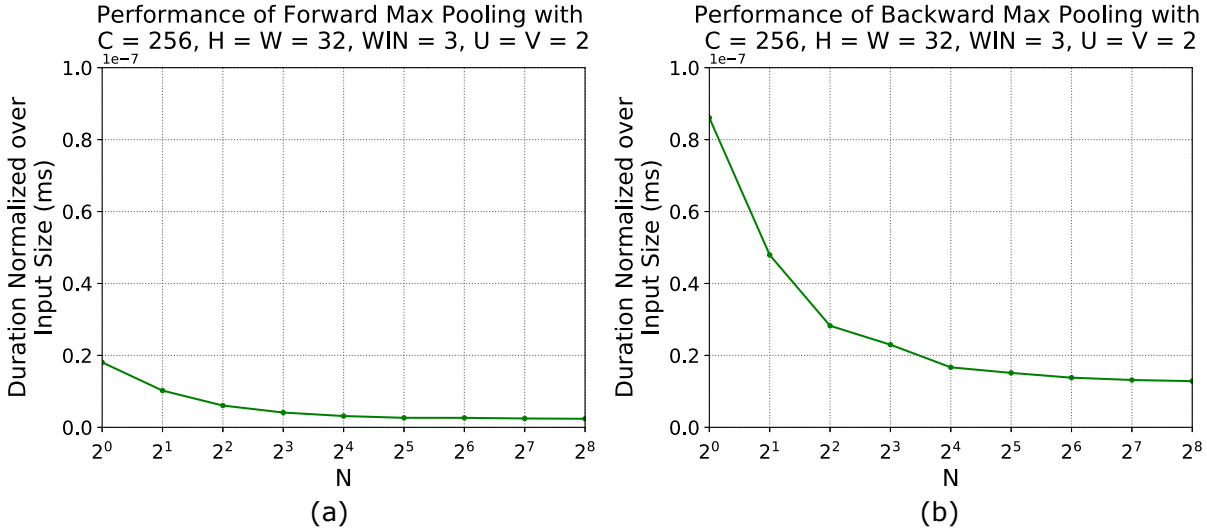


These functions involve few enough calculations that their forward and backward propagation speed is dictated by memory bandwidth. Sigmoid, ReLU, and tanh functions all rely only on the individual activation value, and so have very similar memory access requirements and thus performance. Figure 3 shows that duration is consistently proportional to the number of activations (here, `N*H*W*C`). Consequently, reducing the number of activations is the only way to speed up activation functions. As an exception to this rule, very small activation tensors may not be transferring enough data to and from memory to saturate bandwidth; this behavior is visible for the smallest batch sizes in the backward activation chart for the `H=W=32` case shown in Figure 3.

Chapter 5. Pooling

Pooling layers are commonly used in neural networks to introduce robustness to small spatial variations in the input, and to reduce the spatial dimensions (height and width) of the activations flowing through the neural network. They are defined by the dimensions of the input (`N x H x W x C`), the type of pooling (for example, maximum or average across the activations inside the pooling window), the size and shape of the pooling window (`win_h` and `win_w`), and stride between applications of the pooling window (U and V).

Figure 4. Duration becomes proportional to the input size for larger dimensions. Performance differs for forward and backward propagation of pooling operations. Note that duration is normalized over N*H*W*C here. NVIDIA A100-SXM4-80GB, CUDA 11.2, cuDNN 8.1.



Most pooling operations practically used in deep neural networks are memory bound, as they do not perform enough computation per element to amortize the cost of reading the input and writing the output: Even in an ideal (forward pass) implementation, each element is re-used only `win_h * win_w` times. Hence, most practical implementations focus on maximizing the achieved memory bandwidth utilization, and both implementations are shown in Figure 4 to reach bandwidth saturation for large-enough inputs. Once saturation is achieved, execution time is directly proportional to the size of the input tensor.

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