Abstract

Neural network frameworks such as PyTorch and TensorFlow are the workhorses of numerous machine learning applications ranging from object recognition to machine translation. While these frameworks are versatile and straightforward to use, the training of and inference in deep neural networks is resource (energy, compute, and memory) intensive.

In contrast to recent works focusing on algorithmic enhancements, we introduce BrainSlug, a framework that transparently accelerates neural network workloads by changing the default layer-by-layer processing to a depth-first approach, reducing the amount of data required by the computations and thus improving the performance of the available hardware caches. BrainSlug achieves performance improvements of up to 41.1% on CPUs and 35.7% on GPUs. These optimizations come at zero cost to the user as they do not require hardware changes and only need tiny adjustments to the software.

1. INTRODUCTION

Artificial intelligence, and neural networks in particular, have gained immense notoriety in the past few years. Their flexibility means that they can be applied to a wide range of applications, from recommendation systems for online stores, to autonomous driving, to financial fraud detection or optimization of production lines.

One of the main issues with neural networks is their high computational cost. While over the years many algorithmic and hardware optimizations to reduce the cost of these computations been proposed, the sequential way in which a neural network is processed, taking input data and operating on it from layer to layer before moving onto the next input data, has remained largely unchanged. Even though this breadth-first processing is straightforward, the fact that each pass through the network acts on a relatively large amount of data causes constant cache trashing (whether on CPUs, GPUs, or other hardware), reducing their effectiveness and ultimately increasing computation time. This partly explains why processors with extremely high memory bandwidths are used for neural networks so that the processors are never idle.

In this extended abstract we propose the use of a depth-first approach: we take a subset of the input data (e.g., a part of an image) that can fit in L1 cache and compute a set of aggregated layers, then repeat the process for the next subset of the data. At this high level the process sounds simple; however, there are two main issues. First, only certain operations (i.e., layer types) are able to function when given only a subset of the data. Second, processing data in this way requires the user to write specialized compute kernels for each possible sequence of layers. This is clearly difficult to do by hand and points to the need of an automated system to carry this out.

We implemented BrainSlug, a system that enables depth-first computation of neural networks, providing transparent acceleration through improvements to data locality.

2. BRAINSUG: ARCHITECTURE AND IMPLEMENTATION

One of the explicit goals of BrainSlug is to transparently accelerate neural networks (NN) irrespective of the framework (e.g., PyTorch, Theano, Caffe) they are implemented in. Furthermore, we want the acceleration to apply to a wide range of hardware devices including GPUs, CPUs, FPGAs, and vector processors, among others; this is possible because even though their architectures may vary widely, they all rely on a memory hierarchy to speed up memory accesses, precisely the hardware feature that BrainSlug targets.

To comply with these requirements, the BrainSlug architecture introduces the notion of front-ends to support different NN frameworks, and back-ends to be able to execute on different kinds of hardware (see Figure 1).

The BrainSlug front-ends are specific to a particular framework. They are in charge of parsing the NN in whatever format it is in, and passing the structure to the optimizer and returning an optimized model to the user. Moreover, the front-ends provide glue to in-
to further increase in the future. In future we want to
main memory on GPUs, we expect training batch sizes
during training [7] and the generally growing size of
results insights into the benefits of increasing the batch
size during training [7] and the generally growing size of
main memory on GPUs, we expect training batch sizes
to further increase in the future. In future we want to
enhance BrainSlug by adding more front- and back-
ends, enable training, adding more supported layers and
applying more optimizations. We refer the reader to
our technical report\textsuperscript{1} and our project page\textsuperscript{2} for more
details.

\textsuperscript{1}https://arxiv.org/abs/1804.08378
\textsuperscript{2}https://brainslug.info

3. EVALUATION

To evaluate BrainSlug we chose the TorchVision [5]
package. TorchVision contains a series of broadly used
neural network architectures for computer vision ap-
lications. We use the entire set of available networks
ranging from AlexNet [4], DenseNet [2], Inception v3 [8],
ResNet [1], SqueezeNet [3], and VGG [6], a total of 21
different architecture and parameter combinations.

We run all tests on a server with an Intel Xeon E5-
2690v4, an NVIDIA GeForce GTX 1080 Ti, Debian 9,
NVIDIA GPU driver v384.81, CUDA v9.0, ISPC v1.9.2,
Python v3.5.3 and PyTorch v0.3.0 (using cuDNN). We
perform the test times ten times for the GPU and five
times for the CPU and we take the minimum execution
time for both PyTorch and BrainSlug results.

The current version of BrainSlug can accelerate
Sigmoid, ReLU, Threshold, Batch Normalization, Avg-
and Max-Pooling layers. All other layers use the default
implementation of the used framework.

Table 1 shows BrainSlug’s speed-up results for all
networks on CPU and GPU for batch sizes from 1 to
256. The results clearly indicate that BrainSlug out-
performs PyTorch on the GPU with batch sizes bigger
than 8 (except for ResNet-101 and -152), and for all
network architectures on the CPU.

While the networks have significantly varying execu-
tion times ranging from very short (AlexNet) to quite
long (DenseNet and ResNet), BrainSlug provides a
speed up in all cases, with the most pronounced im-
provements for DenseNets on both CPU and GPU, VGGs
with Batch Normalization on GPU, and SqueezeNets
on CPU. Adding the Batch Normalization layer to the
VGG networks has a significant impact on PyTorch’s
computation time, while there is virtually no change
in BrainSlug’s case: an effect directly attributable to
BrainSlug merging the normalization into the previ-
ous layer. Therefore the speed up is significantly
higher for the VGG-BN network architectures.

Finally, note that negative values for the GPU batch
sizes 1–4 look significant but in absolute terms they are
not: in these cases the execution time is only a few
milliseconds, while for larger batch sizes, it is hundreds
of milliseconds. This relatively performance difference
is mainly due to our implementation being optimized
for larger batch sizes.

4. DISCUSSION AND FUTURE WORK

We have shown that BrainSlug accelerates com-
monly used deep neural networks by as much as 41.1%
on CPUs and 35.7% on GPUs while requiring minimal
code changes. These improvements are significant con-
sidering that training such networks on big data can
take up to several weeks. BrainSlug’s speed-up is
most pronounced for the more commonly used batch
sizes of 8 and up. For instance, the DenseNet architec-
tures are usually trained with a per-GPU batch size of
at least 32 [2], a batch size where BrainSlug achieves
the best performance improvement. Due to recent re-
sults insights into the benefits of increasing the batch
size during training [7] and the generally growing size of
main memory on GPUs, we expect training batch sizes
to further increase in the future. In future we want to
enhance BrainSlug by adding more front- and back-
ends, enable training, adding more supported layers and
applying more optimizations. We refer the reader to
our technical report\textsuperscript{1} and our project page\textsuperscript{2} for more
details.
Table 1: BrainSlug full speed-up results compared to PyTorch on a CPU and GPU for all neural networks.

5. REFERENCES


