BrainSlug: Transparent Acceleration of Deep Learning Through Depth-First Parallelism

Nicolas Weber, Florian Schmidt, Mathias Niepert, Felipe Huici
NEC Laboratories Europe, Systems and Machine Learning Group

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. BRAINSLUG: 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-

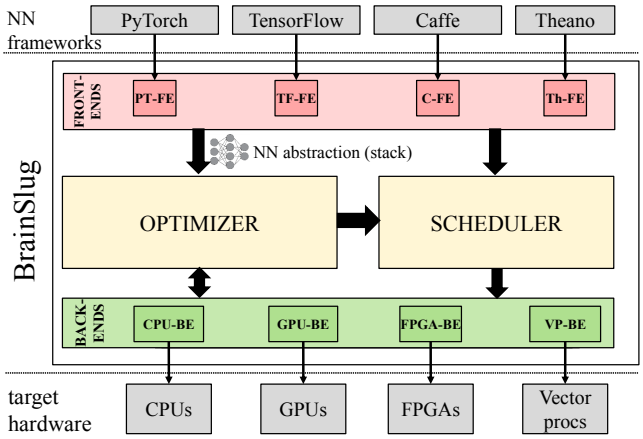

Figure 1: BrainSlug architecture consiting of front-ends (FE) that plug to existing frameworks and convert their NNs into a common abstraction; the optimizer that generates the code to transparently accelerate them; and the scheduler that executes such code, relying on the back-ends (BE) to run it on different target hardware.

```
1 import torchvision.models as models
2 import brainslug
3 model = models.__dict__['...']()
4 brainslug.optimize(model)
5 model(...)
```

Listing 1: Snippet showing how to use BrainSlug's PyTorch front-end. Only lines 2 and 8 need to be added by the user.

voke BrainSlug's scheduler component whenever the framework launches the prediction process. The back-ends provide the necessary glue to have BrainSlug-generated code execute on different kinds of hardware, including providing hardware specs to the optimizer component to help it in generating the code. Listing 1 shows an example for using brainslug with PyTorch.

3. EVALUATION

To evaluate BrainSlug we chose the TorchVision [5] package. TorchVision contains a series of broadly used neural network architectures for computer vision applications. 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 outperforms 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 execution times ranging from very short (AlexNet) to quite long (DenseNet and ResNet), BrainSlug provides a speed up in all cases, with the most pronounced improvements 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 previous 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 commonly 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 considering 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 architectures 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 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[1] and our project page[2] for more details.

[1] https://arxiv.org/abs/1804.08378
[2] https://brainslug.info

GPU (NVIDIA Geforce GTX 1080 Ti)

	1	2	4	8	16	32	64	128	256
AlexNet	-3.0%	-2.9%	-0.3%	1.7%	7.7%	**11.5%**	6.2%	6.8%	7.5%
Inception V3	5.1%	3.0%	4.5%	5.6%	8.1%	**9.4%**	8.4%	7.9%	7.0%
Densenet-121	-9.5%	5.4%	6.3%	22.3%	29.2%	**34.0%**	32.6%	29.6%	24.9%
Densenet-161	-0.2%	3.6%	11.9%	17.8%	24.8%	24.8%	**27.4%**	21.7%	16.0%
Densenet-169	5.4%	7.1%	7.1%	18.8%	26.4%	**32.5%**	30.8%	27.8%	21.9%
Densenet-201	-0.8%	-0.1%	19.0%	21.6%	31.7%	**35.7%**	29.3%	25.4%	18.5%
Resnet-18	-51.6%	-42.7%	-25.8%	6.6%	20.6%	**26.9%**	22.7%	17.2%	13.1%
Resnet-34	-43.1%	-30.6%	0.1%	**23.6%**	20.9%	20.8%	17.6%	9.4%	9.0%
Resnet-50	-27.1%	-8.5%	6.2%	7.5%	8.8%	**10.5%**	7.7%	7.4%	4.7%
Resnet-101	-3.6%	-2.3%	-5.2%	-3.8%	-2.3%	-2.4%	**6.2%**	3.2%	1.7%
Resnet-152	0.4%	-2.8%	-5.8%	-3.9%	-2.0%	-2.5%	**5.9%**	3.6%	1.1%
Squeezenet 1.0	-54.0%	-44.3%	-28.5%	11.6%	12.8%	16.1%	**17.5%**	17.0%	16.9%
Squeezenet 1.1	-27.8%	-26.0%	-20.6%	-0.9%	7.8%	12.8%	**16.1%**	15.8%	15.5%
VGG-11	2.8%	2.2%	5.1%	12.3%	10.8%	14.9%	14.1%	**16.6%**	16.4%
VGG-11 BN	8.7%	6.0%	11.1%	24.4%	26.0%	26.8%	26.7%	29.4%	**30.5%**
VGG-13	0.5%	2.2%	4.6%	8.7%	7.7%	9.4%	9.5%	10.7%	**11.0%**
VGG-13 BN	16.3%	8.2%	12.3%	22.2%	23.8%	23.3%	23.5%	24.3%	**24.8%**
VGG-16	3.8%	-0.7%	0.7%	7.7%	7.5%	8.3%	8.4%	8.7%	**8.8%**
VGG-16 BN	12.4%	7.4%	11.5%	18.8%	19.5%	18.2%	19.9%	21.4%	**22.1%**
VGG-19	4.2%	2.2%	4.1%	7.9%	5.2%	4.4%	6.7%	8.0%	**8.5%**
VGG-19 BN	9.6%	7.4%	11.0%	14.5%	16.6%	14.9%	17.7%	19.2%	**20.0%**

CPU (Intel Xeon E5-2690 v4)

1	2	4	8	16	32	64	128	256
5.3%	4.5%	2.4%	2.4%	1.9%	2.5%	2.7%	2.7%	3.1%
6.9%	6.9%	6.8%	6.2%	4.8%	6.3%	5.2%	6.9%	**7.3%**
15.2%	15.2%	14.9%	11.9%	11.9%	11.8%	10.2%	12.0%	11.9%
16.8%	13.5%	12.2%	11.6%	10.4%	11.2%	10.1%	9.8%	9.5%
19.1%	18.7%	16.3%	13.6%	12.3%	12.8%	12.8%	11.7%	11.2%
23.2%	19.9%	11.7%	14.6%	14.4%	13.6%	11.8%	11.4%	12.8%
6.4%	3.7%	3.9%	0.5%	3.5%	4.0%	3.7%	3.8%	3.5%
3.0%	1.7%	1.9%	**3.5%**	2.9%	2.9%	2.3%	2.6%	2.6%
6.7%	4.5%	3.4%	2.2%	2.7%	2.7%	2.8%	3.4%	3.5%
4.7%	2.7%	2.5%	2.6%	2.3%	2.0%	2.7%	2.7%	3.1%
5.5%	3.3%	2.2%	1.4%	0.5%	1.3%	1.7%	2.6%	2.7%
41.1%	31.4%	26.6%	22.8%	26.9%	29.1%	24.1%	20.4%	20.3%
39.7%	26.2%	22.3%	17.8%	21.1%	32.1%	20.6%	20.0%	21.2%
11.1%	**13.2%**	10.4%	7.0%	6.5%	5.7%	2.6%	2.5%	2.5%
12.4%	**13.1%**	10.6%	8.2%	7.3%	6.7%	3.8%	3.7%	3.7%
11.8%	9.4%	6.4%	5.0%	4.5%	4.1%	2.4%	2.3%	2.3%
8.3%	8.2%	8.0%	5.6%	5.2%	5.1%	3.3%	3.2%	3.2%
6.2%	4.4%	5.2%	4.6%	4.1%	3.8%	1.7%	1.7%	1.7%
6.0%	**7.3%**	7.2%	5.2%	4.8%	4.5%	2.4%	2.4%	2.4%
11.9%	7.7%	5.5%	4.9%	4.0%	3.7%	1.3%	1.2%	1.2%
5.1%	4.9%	4.9%	4.7%	4.2%	4.2%	1.9%	1.9%	1.9%

Table 1: BrainSlug full speed-up results compared to PyTorch on a CPU and GPU for all neural networks.

5. REFERENCES

[1] He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2016), pp. 770–778.

[2] Huang, G., Liu, Z., Weinberger, K. Q., and van der Maaten, L. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2017), vol. 1, p. 3.

[3] Iandola, F. N., Han, S., Moskewicz, M. W., Ashraf, K., Dally, W. J., and Keutzer, K. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <0.5mb model size. *arXiv:1602.07360* (2016).

[4] Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In *Proceedings of Neural Information Processing Systems* (2012).

[5] PyTorch core team. TorchVision. `https://github.com/pytorch/vision`.

[6] Simonyan, K., and Zisserman, A. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).

[7] Smith, S. L., Kindermans, P.-J., and Le, Q. V. Don't decay the learning rate, increase the batch size. In *International Conference on Learning Representations (ICLR)* (2018).

[8] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. Rethinking the inception architecture for computer vision. *CoRR abs/1512.00567* (2015).