Journal of Machine Learning Research 1 (2011) 1-48 Submitted 4/00; Published 10/00

Theano: Deep Learning on GPUs with Python

James Bergstra BERGSTRJ@IRO.UMONTREAL.CA
Frédéric Bastien BASTIENF@IRO.UMONTREAL.CA
Olivier Breuleux BREULEUO@IRO.UMONTREAL.CA
Pascal Lamblin LAMBLINP@IRO.UMONTREAL.CA
Razvan Pascanu PASCANUR@IRO.UMONTREAL.CA
Olivier Delalleau DELALLEA@IRO.UMONTREAL.CA
Guillaume Desjardins DESJAGUI@IRO.UMONTREAL.CA
David Warde-Farley WARDEFAR@IRO.UMONTREAL.CA
Ian Goodfellow GOODFELI@IRO.UMONTREAL.CA
Arnaud Bergeron BERGEARN@IRO.UMONTREAL.CA
Yoshua Bengio BENGIOY@IRO.UMONTREAL.CA
Département d'Informatique et Recherche Opérationnelle
Université de Montréal
2920 Chemin de la tour
Montréal, Québec, Canada, H3T 1J8

Editor: Leslie Pack Kaelbling

Abstract

In this paper, we present *Theano*[1], a framework in the Python programming language for defining, optimizing and evaluating expressions involving high-level operations on tensors. Theano offers most of NumPy's functionality, but adds automatic symbolic differentiation, GPU support, and faster expression evaluation. Theano is a general mathematical tool, but it was developed with the goal of facilitating research in deep learning. The *Deep Learning Tutorials*[2] introduce recent advances in deep learning, and showcase how Theano makes such algorithms compact, elegant, and fast.

Keywords: compiler, computer algebra system, GPU, symbolic differentiation, Python, convolutional networks, deep learning

1. Overview

Theano is a Python library that aims to improve both execution time and development time of machine learning applications, especially deep learning algorithms. It is a symbolic manipulation engine geared towards optimizing and executing expression graphs on tensors. The core of many machine learning applications is the repeated computation of a complex mathematical expression. Often that expression can be written simply in terms of matrix or tensor operations, even if direct computation of those operations is not the

1. Theano: `http://www.deeplearning.net/software/theano/`
2. Deep Learning Tutorials: `http://www.deeplearning.net/tutorial/`

best strategy for computing the entire expression. Theano provides a high-level description language for such mathematical expressions, and a compiler that uses tricks of the trade, heuristics, auxiliary libraries (when present), and even a Graphical Processing Unit (GPU) device (when present) to evaluate such mathematical expressions as quickly and accurately as possible. Theano makes it easier to leverage the rapid development pattern encouraged by the Python language, to describe mathematical algorithms as you would in a functional language, and to profit from very fast code. Theano is free open source software, distributed under a BSD license. This paper introduces Theano via a simple example (logistic regression), and describes the content and scope of the Deep Learning Tutorials, which show how Theano can be used for deep learning (Bengio, 2009).

2. Theano by Example: Logistic Regression

There are four steps to using Theano in a Python program:

1. Declare symbolic input variables.
2. Construct a symbolic expression graph.
3. Compile one or more *functions* to evaluate particular expressions.
4. Call those functions to evaluate expressions for particular input values.

Figure 1 demonstrates these four steps in a simple program implementing logistic regression.

Declare symbolic input variables: Variables `input` and `target` are defined as purely symbolic double-precision vector and integer scalar respectively. Being "purely symbolic" means that one will need to provide their numeric value when time comes to evaluate expressions they are involved in. This contrast with variables `W` and `b`, defined as *shared* variables, which are associated to a specific numeric value (a NumPy array). Shared variables are managed by Theano, and may automatically be allocated on a GPU. Note that although dense tensor variables enjoy the broadest support, sparse matrices and generic objects are also supported in Theano.

Construct a symbolic graph: The functions `dot`, `softmax`, `log` and others construct an expression graph rooted at our input variables. The variables `probs`, `pred`, `nll`, and `dW` are nodes in that graph. The set of operations (*Ops*) available for building expression graphs corresponds roughly to the functionalities provided by NumPy (arithmetic operators, standard mathematical functions, indexing, broadcasting) and includes some operations specifically useful for deep learning such as `conv2d` and `softmax`. The `grad` function produces a symbolic expression of the cost gradient on parameters `W` and `b` using a symbolic differentiation. For a complete list of supported operations, we refer the reader to the "Library Documentation" available on the Theano website.

Compile functions: Calls to `function` produce callable Python objects that compute certain outputs from the required inputs, with the possible side-effect of updating shared variables. This update mechanism makes it natural to express iterative algorithms such as gradient descent, and can avoid needless transfers between the GPU and its host. By having access to the entire computational graph, Theano can perform the necessary transformations to speed up computation (e.g. simplification, specialization, loop fusion, constant

```
# DECLARE INPUTS
n_in, n_out = 784, 10      # MNIST-sized
input, target = dvector(), iscalar()
W = shared(numpy.zeros((n_in, n_out)))
b = shared(numpy.zeros(n_out)

# DEFINE THE GRAPH
probs = softmax(dot(input, W) + b)
pred = argmax(probs)
nll = -log(probs)[
    arange(target.shape[0]),
    target]
dW, db = grad(nll, [W, b])

# COMPILE THE GRAPH
test = function([input], pred)
train = function([input, target], nll,
    updates = {W: W - 0.1 * dW,
               b: b - 0.1 * db})

# <CALL THE FUNCTIONS>
```

Figure 1: Sample usage of Theano: logistic regression with fixed learning rate. Model parameters `W` and `b` can be allocated on a GPU. The `grad` function hides the details of differentiation.

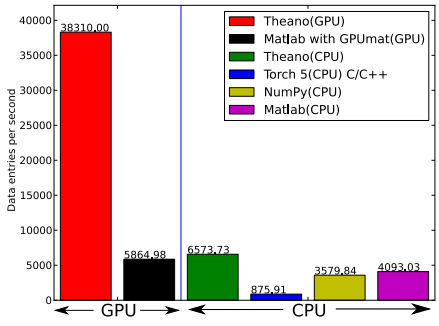

Figure 2: Training speed for a 3-layer DBN (fine-tuning phase) of Theano and various other libraries. Theano is fast in both CPU and GPU compute devices. Additional benchmarking is detailed in Bergstra et al. (2010).

folding, memory reuse, BLAS utilization), and generate custom C++ and CUDA code on both CPU and GPU. Theano can also infer shape and type information allowing it to generate specialized expression implementations. This meta-programming approach is all but essential for obtaining good GPU performance in a wide variety of problem sizes. Theano provides several compilation modes to make different tradeoffs between compilation time and execution time, to perform space and time profiling, and to perform self-verification.

Call functions: The functions `train` and `test` are normal callable Python objects that accept NumPy arguments and return NumPy results. They are fully interoperable with NumPy, even when internal computations are performed on a GPU. The speed of Theano's compiled functions is shown in Figures 2 and 3, which compare gradient descent in DBN and convolutional architectures across several implementations.[3]

3. Deep Learning Tutorials

The Deep Learning Tutorials introduce the advanced undergraduate or graduate computer science student to deep learning algorithms: Logistic Regression, the Multilayer Perceptron (Rumelhart et al., 1986), Convolutional Network (LeCun et al., 1998), Auto-Associator (Rumelhart et al., 1986; Bengio, 2009), Stacked Denoising Auto-Associator (Vincent et al.,

3. Benchmarking code: `http://github.com/pascanur/DeepLearningBenchmarks`
Torch 5: `http://torch5.sourceforge.net`
EBLearn (Energy Based Learning): `http://eblearn.sourceforge.net/`
GPUmat (GPU toolbox for MATLAB): `http://gp-you.org`
CPU timing was carried out on an a Intel(R) Core(TM)2 Duo CPU E8500 @ 3.16GHz using Intel MKL BLAS (one thread). GPU timing was done on a GForce GTX 285.

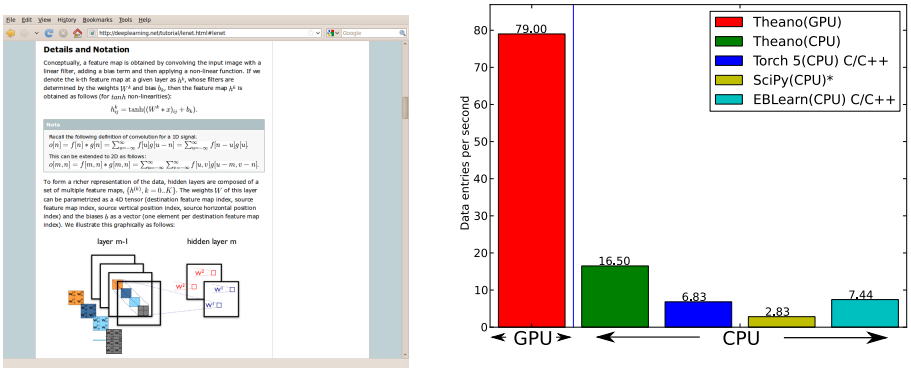

Figure 3: Left: Screenshot from Deep Learning Tutorials page on Convolutional Neural Networks (CNNs). Right: Performance Comparisons between CNN implementations. The benchmark stressed convolution of medium-sized images (256x256) with small (7x7) filters.

2008), Restricted Boltzmann Machine, and Deep Belief Network (Hinton et al., 2006). The tutorials outline the intuition and mathematics for each model, and provide compact implementations based on highly-documented Theano code. Future tutorials are planned on the topics of sparse coding and quadratic energy models.

4. Acknowledgements

The authors would like to thank all Theano contributors for their time and effort.[4] This work was supported by funding from Compute Canada, RQCHP, NSERC, and Canada Research Chairs.

References

Y. Bengio. Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, 2(1): 1–127, 2009. Also published as a book. Now Publishers, 2009.

J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, and Y. Bengio. Theano: a CPU and GPU math expression compiler. In *Proc. SciPy*, Austin, TX, 2010.

G. E. Hinton, S. Osindero, and Y. W. Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18:1527–1554, 2006.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient based learning applied to document recognition. *IEEE*, 86(11):2278–2324, November 1998.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

P. Vincent, H. Larochelle, Y. Bengio, and P. Manzagol. Extracting and composing robust features with denoising autoencoders. In *ICML'08*, pages 1096–1103. ACM, 2008.

4. List of contributors: `http://www.deeplearning.net/software/theano/contributors.html`