Deep Learning on GPUs with Theano

James Bergstra bergstrj@iro.umontreal.ca
Frédéric Bastien bastienf@iro.umontreal.ca
Joseph Turian turian@iro.umontreal.ca
Razvan Pascanu pascanur@iro.umontreal.ca
Olivier Delalleau delallea@iro.umontreal.ca
Olivier Breuleux breuleuo@iro.umontreal.ca
Pascal Lamblin lamblinp@iro.umontreal.ca
Guillaume Desjardins desjagui@iro.umontreal.ca
Dumitru Erhan erhandum@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
http://www-etud.iro.umontreal.ca/∼bergstrj
http://www.deeplearning.net

Since the introduction of Deep Belief Networks (Hinton et al., 2006), a surge of deep learning approaches have achieved state-of-the-art performance in natural language tasks, audio classification, and image classification and demonstrated the advantage of using graphics hardware (GPUs) for computation (Raina et al., 2009; Pinto et al., 2009). But deep learning algorithms are often considered too slow, and implementing them can be difficult. We present the *Theano*[1] library and a set of online tutorials[2] that make research in deep learning more accessible.

Theano is a symbolic companion to Python's `numpy` for dealing with multidimensional arrays of numbers. Computations can be expressed symbolically in Theano, and then compiled by Theano's C and GPU code generators into dynamically loaded modules. Theano optimizes symbolic graphs for numerical stability as well as efficient execution. Theano supports automatic differentiation of graphs as well as GPU execution. Theano's feature set is sufficient to implement state of the art deep learning algorithms (including convolutional algorithms) and run them efficiently.

- *tight integration with numpy*: The `numpy.ndarray` type is accepted directly as a function input, returned as a function output, and used internally for CPU calculations.
- *transparent use of a GPU*: Expression graphs specify computations abstractly, so that it is invisible to the user whether calculations are carried out on the CPU or GPU. The only catch is that large frequently-used data (e.g. model parameters) must be declared to Theano, so that Theano can choose where to allocate them. Calculations are up to 140x faster on GPU than CPU.
- *symbolic differentiation*: Theano can derive gradient expressions automatically, which is very convenient for expressing gradient-based learning algorithms.
- *speed and stability optimizations*: Prior to code generation, Theano rearranges the expression that you requested to compile. Firstly it copies your graph. Then it puts the graph as much as possible into a canonical form. Numerically unstable computations (e.g. $\log(1+x)$) are replaced with stable equivalents, and many special cases are optimized for speed (e.g. $(1)x^1 + 0$).
- *dynamic C code generation*: Many of the expressions offered by Theano are implemented both in Python and C. These C implementations are instantiated and specialized for particular inputs when you compile a Theano function, so they are quite fast. A cache system avoids unnecessary recompilation.
- *extensive testing and self-verification*: Thousands of unit tests are executed every night to catch any regression errors that might be introduced. Theano includes a self-verifying execution mode that detects and diagnoses many kinds of internal error.

Starting from a program using `numpy`, porting code to Theano is often a matter of minutes or hours. Theano can bring performance gains of up to 3-fold using a CPU, or up to 100-fold or more using a GPU.

[1] **Theano:** http://www.deeplearning.net/software/theano
[2] **Deep Learning Tutorials:** http://www.deeplearning.net/tutorials

```
### declare symbolic inputs and parameters
### if Theano is in GPU-mode, w and b will be allocated on the GPU device
x = matrix(); y = vector(dtype='int32')
w = shared(numpy.random.randn(n_input, n_outputs))
b = shared(numpy.zeros(n_outputs))

### symbolic expression of logistic regression
prediction = argmax(softmax(dot(x,w)+b))
loss = -mean(log(softmax(dot(x,w)+b))[arange(y.shape[0]), y])

### compile a training function for stochastic gradient descent
gw, gb = grad(loss, [w, b])
f = function(inputs=[x,y], output=[prediction,loss], updates={w:w-0.1*gw, b:b-0.1*gb}))
```

Figure 1: Theano logistic regression code (with `theano` namespace omitted). The function `f` that is compiled accepts two `numpy.ndarray` arguments `x` and `y`, and returns the model's prediction and its training loss. Theano optimizes this graph in many ways before compiling dynamically-generated C code. For example, it computes `dot(x,w)` just once, it uses a numerically stable algorithm for the logarithm of the softmax, and it avoids several intermediate computations by updating `w` with BLAS's dgemm function. Theano knows that the `softmax` does not change the `argmax` so in the end there is no `softmax` left in the compiled function at all. This example is based on the logistic regression tutorial at `http://www.deeplearning.net/tutorial/logreg.html`.

Deep learning algorithms are dominated by matrix multiplications and convolutions, and gains of 10x to 30x using the GPU are common. Theano is under active development, but it has powered our own experiments in deep machine learning since 2008, it has provided the basis for a GPU-powered machine learning course at the University of Montreal (IFT6266), and it has provided an efficient language for the Deep Learning Tutorials. Theano makes it possible to reproduce and extend the inspiring results and performance of Pinto et al. and Raina et al. on GPUs using short, readable, extensible Python programs – even for users with no idea of how the GPU implementations work.

> "... there are a number of groups who work in computer algebra who may find Theano to offer an attractive amount of abstraction for their work... We would definitely consider suggesting this to users who are keen on doing GPGPU using python."
>
> – Hugh Merz, SHARCNET Analyst in High Performance Computing Accelerators, Astrophysics

The Deep Learning Tutorials (see: `www.deeplearning.net`) introduce a graduate-level student with some knowledge of Python and machine learning to the theory and practice of deep learning. Each tutorial introduces theory and code (based on Theano) for a learning algorithm. To date, the tutorials cover logistic regression, multilayer perceptron, convolutional networks, restricted boltzmann machines, [denoising] autoassociators, deep belief networks, and stacked denoising autoencoders. The tutorials are part of the course material for graduate course IFT6266 at the University of Montreal, and form a practical complement to (Bengio, 2009).

References

Bengio, Y. (2009). Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, *2*.

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

Pinto, N., Doukhan, D., DiCarlo, J. J., & Cox, D. D. (2009). A high-throughput screening approach to discovering good forms of biologically-inspired visual representation. *PLoS Comp. Bio.*, *5*.

Raina, R., Madhavan, A., & Ng, A. Y. (2009). Large-scale deep unsupervised learning using graphics processors. *Proc. of International Conference on Machine Learning (ICML)*.