Theano, Pylearn2, libgpuarray Presentation

Frédéric Bastien, Bart van Merriënboer
Département d'Informatique et de Recherche Opérationnelle
Université de Montréal
Montréal, Canada
`{bastienf, vanmerb}@iro.umontreal.ca`

OML Workshop 2014

Laboratoire d'Informatique
des Systèmes Adaptatifs
http://www.iro.umontreal.ca/~lisa

Université
de Montréal

High level

Python <- {NumPy/SciPy/libgpuarray} <- Theano <- Pylearn2

- Python: OO coding language
- Numpy: *n*-dimensional array object and scientific computing toolbox
- SciPy: sparse matrix objects and more scientific computing functionality
- libgpuarray: GPU *n*-dimensional array object in C for CUDA and OpenCL
- Theano: compiler/symbolic graph manipulation
- Pylearn2: machine learning framework

Python

- General-purpose high-level OO interpreted language
- Emphasizes code readability
- Comprehensive standard library
- Dynamic type and memory management
- Slow execution
- Easily extensible with C
- Popular in *web development* and *scientific communities*

NumPy/SciPy

- Python floats are full-fledged objects on the heap
    - Not suitable for high-performance computing!
- NumPy provides an *n*-dimensional numeric array in Python
    - Perfect for high-performance computing
    - Slices of arrays are views (no copying)
- NumPy provides
    - Elementwise computations
    - Linear algebra, Fourier transforms
    - Pseudorandom number generators (many distributions)
- SciPy provides lots more, including
    - Sparse matrices
    - More linear algebra
    - Solvers and optimization algorithms
    - Matlab-compatible I/O
    - I/O and signal processing for images and audio

What's missing?

- Non-lazy evaluation (required by Python) hurts performance
- Bound to the CPU
- Lacks symbolic or automatic differentiation
- No automatic speed and stability optimization

Theano

High-level domain-specific language tailored to numeric computation.

- Syntax as close to NumPy as possible
- Compiles most common expressions to C for CPU and/or GPU
- Limited expressivity means more opportunities optimizations
    - No subroutines -> global optimization
    - Strongly typed -> compiles to C
    - Array oriented -> easy parallelism
    - Support for looping and branching in expressions
- Automatic speed and stability optimizations
- Can reuse other technologies for best performance.
    - BLAS, SciPy, Cython, Numba, PyCUDA, CUDA
- Automatic differentiation and R op
- Sparse matrices

Pylearn2

Machine Learning library aimed at researchers

- Built on top of Theano, for fast execution and use of GPU
- Easy to try variants of implemented algorithms, and to extend them (using Theano)
- Very modular, each component of the library can be used in isolation
- Experiments can be specified through a YAML config file, or by a Python script
- Scripts for visualizing weights, plot monitored values

libgpuarray

Goal: A common GPU *n*-dimensional array that can be reused by all projects, support for both CUDA and OpenCL.

Motivation:

- Currently there are at least 6 different GPU arrays in Python
  - CudaNdarray (Theano), GPUArray (pycuda), CUDAMatrix (cudamat), GPUArray (pyopencl), Clyther, Copperhead, ...
  - There are even more if we include other languages.
- They are incompatible
  - None have the same properties and interface
- All of them implement a subset of numpy.ndarray properties
- This is the new GPU backend on Theano

Goal of the stack

Fast to develop
Fast to run

Introduction

Theano

Pylearn2

libgpuarray

Conclusion

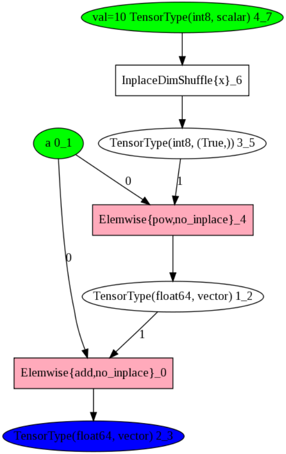
Description

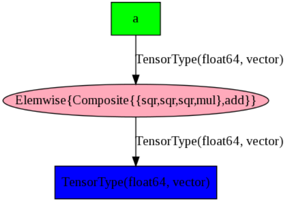
- Mathematical symbolic expression compiler
- Expressions mimic NumPy's syntax and semantics
- Dynamic C/CUDA code generation
  - C/C++, CUDA, OpenCL, PyCUDA, Cython, Numba, …
- Efficient symbolic differentiation
- Speed and stability optimizations
  - Gives the right answer for "$\log(1 + x)$" even if x is really tiny.
- Extensive unit-testing and self-verification
- Works on Linux, OS X and Windows
- Transparent use of a GPU
  - `float32` only for now (libgpuarray provides much more)
  - Limited support on Windows
- Sparse operations (CPU only)

Simple example

```
import theano
# declare symbolic variable
a = theano.tensor.vector("a")
# build symbolic expression
b = a + a ** 10
# compile function
f = theano.function([a], b)
print f([0, 1, 2])
# prints `array([0, 2, 1026])`
```

Simple example: graph optimization


val=10 TensorType(int8, scalar) 4_7
InplaceDimShuffle{x}_6
a 0_1
TensorType(int8, (True,)) 3_5
0
1
Elemwise{pow,no_inplace}_4
0
TensorType(float64, vector) 1_2
1
Elemwise{add,no_inplace}_0
TensorType(float64, vector) 2_3


a
TensorType(float64, vector)
Elemwise{Composite{{sqr,sqr,sqr,mul},add}}
TensorType(float64, vector)
TensorType(float64, vector)

Project status?

- Mature: Theano has been developed and used since January 2008 (6.5 yrs old)
- Driven over 100 research papers
- Good user documentation
- Active mailing list with participants from outside our lab
- Core technology for a few Silicon-Valley start-ups
- Many contributors (some from outside our lab)
- Used to teach many university classes
- Has been used for research at Google and Yahoo.

Theano: `deeplearning.net/software/theano/`
Deep Learning Tutorials: `deeplearning.net/tutorial/`

Pylearn2 details

The core library contains a collection of:

- Training algorithms (e.g. Stochastic and Batch GD, model-specific rules)
  - Costs, supervised/unsupervised and exact/estimated (e.g. NLL, Score matching, NCE)
  - Monitor, history of (functions of) parameters and hyperparameters on different data sets (training, validation, test)
  - Termination criteria, determine when to stop training
- Training extensions, perform actions throughout the training process (e.g., early stopping)
- Models (e.g. NNets, ConvNets, RBMs, k-means, PCA, SVMs)
- Datasets (e.g. MNIST, CIFAR-10) and preprocessors (LCN, ZCA)

Pylearn2 details, continued

- Data specifications which give semantics to data
    - IndexSpace, 1D integer array e.g. for labels
    - VectorSpace, 1D float array e.g. for softmax output
    - Conv2DSpace, 3D float32 arrays e.g. for color image input
- Allows for automatic conversion when needed e.g. labels to one-hot vectors, images to flattened vectors
- YAML file allows experiments to be conducted without writing code

Project status

- Has been used for scientific publications, Kaggle competitions, used by many researchers at LISA
- Still under rapid development, however the API shouldn't break without warning
- Documentation is incomplete, but quickly improving
- Active mailing list with participants from outside our lab
- Core technology for a least one Silicon-Valley start-up
- Features currently in development:
  - Recurrent neural networks (RNNs), based on the GroundHog framework developed at LISA
  - Better hyperparameter search support, using e.g. Hyperopt

libgpuarray: Design Goals

- Have the base object in C to allow collaboration with more projects.
    - We want people from C, C++, ruby, R, . . . all use the same base GPU ndarray.
- Be compatible with CUDA and OpenCL.
- Not too simple, (don't support just matrix).
- Support all dtype.
- Allow strided views.
- But still easy to develop new code that support only a few memory layout.
    - This ease the development of new code.

Project status?

- Usable directly, but not all implementation available.
- Multiple GPUs works.
- Is the next GPU array container for Theano and is working.
  - Not all Theano implementations available now.
  - OpenCL misses more implementations.
  - Multiple GPUs on the way.
- Web site:
  `http://deeplearning.net/software/libgpuarray/`

Conclusion

Theano/Pylearn2/libgpuarry provide an environment for machine learning that is: **Fast to develop**
Fast to run

Acknowledgments

- All people working or having worked at the LISA lab.
- All Theano/Pylearn 2 users/contributors
- Compute Canada, RQCHP, NSERC, and Canada Research Chairs for providing funds or access to compute resources.

Questions?