Theano at a Glance: A Framework for Machine Learning

Frédéric Bastien

Montreal Institute for Learning Algorithms
Université de Montréal
Montréal, Canada
bastienf@iro.umontreal.ca

Presentation prepared with Pascal Lamblin and Simon Lefrancois

MILA

AI With The Best 2016

Université de Montréal

High level

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

- 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
  - **(Not specific to machine learning)**
- {...}: Many libraries built on top of Theano

What Theano provides

- Lazy evaluation for performance
- GPU support
- Symbolic differentiation
- Automatic speed and stability optimization

High level

Many [machine learning] library build on top of Theano

- Keras
- blocks
- lasagne
- sklearn-theano
- PyMC 3
- theano-rnn
- Morb
- ...

Goal of the stack

Fast to develop
Fast to run

Some models build with Theano

Some models that have been build with Theano.

- Neural Networks
- Convolutional Neural Networks, AlexNet, OverFeat, GoogLeNet
- RNN, CTC, LSTM, GRU
- NADE, RNADE, MADE
- Autoencoders
- Generative Adversarial Nets
- SVMs
- **many variations of above models and more**

Project status

- Mature: Theano has been developed and used since January 2008 (8 yrs old)
- Driven hundreds of research papers
- Good user documentation
- Active mailing list with worldwide participants
- Core technology for Silicon-Valley start-ups
- Many contributors (some from outside our institute)
- Used to teach many university classes
- Has been used for research at big compagnies
- Theano 0.8.2 released 21th of April, 2016

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

Theano community

Active community

- Many people reply on our mailing lists
- Hundreds of answered questions on StackOverflow
- 141 contributors to Theano 0.8
- Main developers at MILA

MILA

Institut des algorithmes d'apprentissage de Montréal
Montreal Institute for Machine Learning

- Professors 7
- Staff 7
- Visiting Scientists 1
- Post-Doc 6
- Ph.D. students 42
- M.Sc. students 22
- Interns 22
- Total 107

`mila.umontreal.ca`

formely known as LISA

MILA Partners

- Many universities
- Ubisoft
- Nuance
- IBM
- Google
- Facebook
- NVIDIA
- Huawei
- Intel
- D-Wave
- ApSTAT
- Qualcomm
- Imagia
- Sulfur Heron

Python

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

Description

High-level domain-specific language for 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 for optimizations
  - **Strongly typed** -> compiles to C
  - **Array oriented** -> easy parallelism
  - Support for **looping and branching** in expressions
  - No subroutines -> **global optimization**
- **Automatic** speed and numerical stability **optimizations**

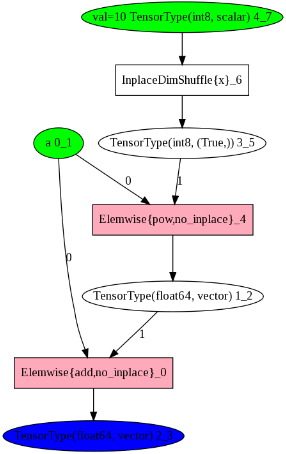
Description (2)

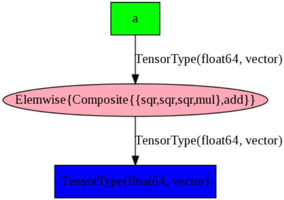
- **Symbolic differentiation and R op** (Hessian Free Optimization)
- Can **reuse other technologies** for best performance
  - CUDA, CuBLAS, CuDNN, BLAS, SciPy, PyCUDA, Cython, Numba, ...
- Sparse matrices (CPU only)
- Extensive unit-testing and self-verification
- Extensible (You can create new operations as needed)
- Works on **Linux, OS X and Windows**
- **Multi-GPU** (via platoon)
- New GPU back-end:
  - **Float16** new back-end (need cuda 7.5)
  - **Multi dtypes**
  - Multi-GPU support in the same process

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)

# Execute with numerical value
print f([0, 1, 2])
# prints `array([0, 2, 1026])`
```

Simple example





Overview of library

Theano is many things

- Language
- Compiler
- Python library

Compiling and running expression

- theano.function
- shared variables and updates
- compilation modes

Shared variable example

```
>>> from theano import shared
>>> x = shared(0.)  # Normally, model parameters
>>> updates = [(x, x + 1)]
>>> f = function([], updates=updates)
>>> f()
>>> x.get_value()
1.0
>>> x.set_value(100.)
>>> f()
>>> x.get_value()
101.0
```

Modifying expressions

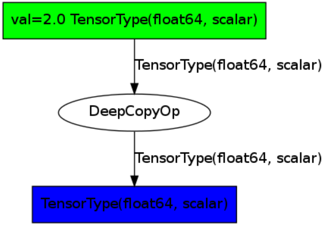
There are “macro” that automatically build bigger graph for you.

- theano.grad
- R_op, L_op for Hessian Free Optimization
- hessian
- jacobian
- clone the graph with replacement
- you can navigate the graph if you need

Those functions can get called many times, for example to get the 2nd derivative.

The grad method

```
>>> x = T.scalar('x')
>>> cost = 2. * x
>>> g = T.grad(cost, x)

# Print the optimized graph
>>> f = theano.function([x], g)
>>> theano.printing.pydotprint(f)
```



Enabling GPU

- Theano's current back-end only supports 32 bit on GPU
- libgpuarray (new-backend) supports all dtype
- CUDA supports float64, but it is slow on gamer GPUs

CuDNN

- V4, V5 and V5.1 are supported.
- It is enabled automatically if available.
- Theano flag to get an error if can't be used: "dnn.enabled=True"
- Theano flag to disable it: "dnn.enabled=False"

Debugging

- DebugMode: a mode that tests many things done by Theano (very slow)
- NanGuardMode: a mode that help find the cause of nan in the graph.
- Error message
- theano.printing.debugprint: print a textual representation of computation

Error message: code

```
import numpy as np
import theano
import theano.tensor as T
x = T.vector()
y = T.vector()
z = x + x
z = z + y
f = theano.function([x, y], z)
f(np.ones((2,)), np.ones((3,)))
```

Error message: 1st part

```
Traceback (most recent call last):
[...]
ValueError: Input dimension mis-match.
    (input[0].shape[0] = 3, input[1].shape[0] = 2)
Apply node that caused the error:
   Elemwise{add,no_inplace}(<TensorType(float64, vector)>,
                            <TensorType(float64, vector)>,
                            <TensorType(float64, vector)>)
Inputs types: [TensorType(float64, vector),
               TensorType(float64, vector),
               TensorType(float64, vector)]
Inputs shapes: [(3,), (2,), (2,)]
Inputs strides: [(8,), (8,), (8,)]
Inputs values: [array([ 1.,  1.,  1.]),
                array([ 1.,  1.]),
                array([ 1.,  1.])]
Outputs clients: [['output']]
```

Backtrace

```
Backtrace when the node is created:
  File "test.py", line 7, in <module>
    z = z + y
```

Error message: 2nd part

HINT: Re-running with most Theano optimization disabled could give you a back-traces when this node was created. This can be done with by setting the Theano flags "optimizer=fast_compile". If that does not work, Theano optimizations can be disabled with "optimizer=None".

HINT: Use the Theano flag "exception_verbosity=high" for a debugprint of this apply node.

debugprint

```
>>> from theano.printing import debugprint
>>> debugprint(a)
Elemwise{mul,no_inplace} [id A] ''
 |TensorConstant{2.0} [id B]
 |Elemwise{add,no_inplace} [id C] 'z'
   |<TensorType(float64, scalar)> [id D]
   |<TensorType(float64, scalar)> [id E]
```

Inputs

```
# Load from disk and put in shared variable.
datasets = load_data(dataset)
train_set_x, train_set_y = datasets[0]
valid_set_x, valid_set_y = datasets[1]

# allocate symbolic variables for the data
index = T.lscalar()  # index to a [mini]batch

# generate symbolic variables for input minibatch
x = T.matrix('x')  # data, 1 row per image
y = T.ivector('y')  # labels
```

Model

```
n_in = 28 * 28
n_out = 10

# weights
W = theano.shared(
        numpy.zeros((n_in, n_out),
                    dtype=theano.config.floatX))

# bias
b = theano.shared(
        numpy.zeros((n_out,),
                    dtype=theano.config.floatX))
```

Computation

```
# the forward pass
p_y_given_x = T.nnet.softmax(T.dot(input, W) + b)

# cost we minimize: the negative log likelihood
l = T.log(p_y_given_x)
cost = -T.mean(l[T.arange(y.shape[0]), y])

# the error
y_pred = T.argmax(p_y_given_x, axis=1)
err = T.mean(T.neq(y_pred, y))
```

Gradient and updates

```
# compute the gradient of cost
g_W, g_b = T.grad(cost=cost, wrt=(W, b))

# model parameters updates rules
updates = [(W, W - learning_rate * g_W),
           (b, b - learning_rate * g_b)]
```

Training function

```
# compile a Theano function that train the model
train_model = theano.function(
    inputs=[index], outputs=(cost, err),
    updates=updates,
    givens={
        x: train_set_x[index * batch_size:
                       (index + 1) * batch_size],
        y: train_set_y[index * batch_size:
                       (index + 1) * batch_size]
    }
)
```

Convolution computation

```
# convolve input feature maps with filters
conv_out = conv.conv2d(input=x, filters=W)

# pool each feature map individually,
# using maxpooling
pooled_out = pool.pool_2d(
    input=conv_out,
    ds=(2, 2),  # poolsize
    ignore_border=True)

output = T.tanh(pooled_out +
                b.dimshuffle('x', 0, 'x', 'x'))
```

Scan

- Allows looping (for, map, while)
- Allows recurrence (reduce)
- Allows recurrence with dependency on many of the previous time steps
- Optimize some cases like moving computation outside of scan
- The Scan grad is done via Backpropagation Through Time (BPTT)

Theano 0.8

Released 21th of March, 2016

Highlights:

- Faster compilation and execution
- Integration of CuDNN for better GPU performance
- Many Scan improvements (execution speed up, ...)
- Interactive visualization of graphs with d3viz
- optimizer=fast_compile moves computation to the GPU.
- Multi-GPU for data parallism via Platoon
  `https://github.com/mila-udem/platoon`
- cnmem (better memory management on GPU)
- New GPU back-end

A total of 141 people contributed to this release.

Since last release

Highlights:

- Computation and compilation speed up
- Multi-cores convolution and pooling on CPU
- More numerical stability by default for some graphs
- CuDNN: 5.1, batch normalization, (RNN soon)
- Dilated convolution
- Multiple-GPU, synchronous update (via platoon)
- Partial function evaluation
- Add gradient of solve, tensorinv (CPU), tensorsolve (CPU) searchsorted (CPU)
- Add Multinomial Without Replacement
- ...

90 people already contributed to the development version.

Roadmap

- Deprecate the old GPU back-end
- Faster compile time
  - First time compilation (less C code compilations, already cached)
  - Of big graph (faster graph optimization)
- Handling of bigger graph
- Speed/Memory trade off
  - Scan very long sequence, (ex: Audio recognition and generation)
  - Recompute cheap operation of the forward during the gradient
- More Multi-GPU data parallelism algo

Where to learn more

- Deep Learning Tutorials with Theano: `deeplearning.net/tutorial`
- Theano tutorial: `deeplearning.net/software/theano/tutorial`
- Theano website: `deeplearning.net/software/theano`
- Doc of frameworks on top of Theano like Blocks, Keras, Lasagne, ...

Questions, acknowledgments

Questions?
Acknowledgments

- All people working or having worked at MILA institute (previously LISA lab).
- All Theano users/contributors.
- Compute Canada, RQCHP, NSERC, NVIDIA, and Canada Research Chairs for providing funds, access to computing resources, hardware or GPU libraries.