Introduction to Theano

Arnaud Bergeron
(slides adapted by Frédéric Bastien from slides by Ian G.)
(further adapted by Arnaud Bergeron)

February 26, 2015

High level

- Overview of library (3 min)
- Building expressions (30 min)
- Compiling and running expressions (30 min)
- Modifying expressions (25 min)
- Debugging (30 min)
- Citing Theano (2 min)

Overview of Library

Theano is many things

- ► Language
- ► Compiler
- ► Python library

Overview

Theano language:

- ▶ Operations on scalar, vector, matrix, tensor, and sparse variables
- ▶ Linear algebra
- ▶ Element-wise nonlinearities
- ▶ Convolution
- ▶ Extensible

Overview

Using Theano:

- define expression $f(x, y) = x + y$

```
>>> z = x + y
```

- compile expression

```
>>> f = theano.function([x, y], z)
```

- execute expression

```
>>> f(1, 2)
3
```

Building expressions

- ▶ Scalars
- ▶ Vectors
- ▶ Matrices
- ▶ Tensors
- ▶ Reduction
- ▶ Dimshuffle

Scalar math

```
from theano import tensor as T
x = T.scalar()
y = T.scalar()
z = x+y
w = z*x
a = T.sqrt(w)
b = T.exp(a)
c = a ** b
d = T.log(c)
```

Vector math

```
from theano import tensor as T
x = T.vector()
y = T.vector()
# Scalar math applied elementwise
a = x * y
# Vector dot product
b = T.dot(x, y)
# Broadcasting
c = a + b
```

Matrix math

```
from theano import tensor as T
x = T.matrix()
y = T.matrix()
a = T.vector()
# Matrix-matrix product
b = T.dot(x, y)
# Matrix-vector product
c = T.dot(x, a)
```

Tensors

- Dimensionality defined by length of "broadcastable" argument
- Can add (or do other elemwise op) on two tensors with same dimensionality
- Duplicate tensors along broadcastable axes to make size match

```python
from theano import tensor as T
tensor3 = T.TensorType(
    broadcastable=(False, False, False),
    dtype='float32')
x = tensor3()
```

Reductions

```
from theano import tensor as T
tensor3 = T.TensorType(
    broadcastable=(False, False, False),
    dtype='float32')
x = tensor3()
total = x.sum()
marginals = x.sum(axis=(0, 2))
mx = x.max(axis=1)
```

Dimshuffle

```python
from theano import tensor as T
tensor3 = T.TensorType(
    broadcastable=(False, False, False),
    dtype='float32')
x = tensor3()
y = x.dimshuffle((2, 1, 0))
a = T.matrix()
b = a.T
# Same as b
c = a.dimshuffle((0, 1))
# Adding to larger tensor
d = a.dimshuffle((0, 1, 'x'))
e = a + d
```

Exercices

Work through the "Building Expressions" section of the ipython notebook.

Compiling and running expression

- ► `theano.function`
- ► shared variables and updates
- ► compilation modes
- ► compilation for GPU
- ► optimizations

theano.function

```
>>> from theano import tensor as T
>>> x = T.scalar()
>>> y = T.scalar()
>>> from theano import function
>>> # first arg is list of symbolic inputs
>>> # second arg is symbolic output
>>> f = function([x, y], x + y)
>>> # Call it with numerical values
>>> # Get a numerical output
>>> f(1., 2.)
array(3.0)
```

Shared variables

- It's hard to do much with purely functional programming
- *shared variables* add just a little bit of imperative programming
- A *shared variable* is a buffer that stores a numerical value for a Theano variable
- Can write to as many shared variables as you want, once each, at the end of the function
- Modify outside Theano function with `get_value()` and `set_value()` methods.

Shared variable example

```
>>> from theano import shared
>>> x = shared(0.)
# Can also use a dict for more complex code
>>> updates = [(x,  x + 1)]
>>> f = function([], updates=updates)
>>> f()
>>> x.get_value()
1.0
>>> x.set_value(100.)
>>> f()
>>> x.get_value()
101.0
```

Which dict?

- ▶ Use theano.compat.python2x.OrderedDict
- ▶ Not collections.OrderedDict
  - ▶ This isn't available in older versions of python, and will limit the portability of your code.
- ▶ Not {} aka dict
  - ▶ The iteration order of this built-in class is not deterministic so if Theano accepted this, the same script could compile different C programs each time you run it.

Compilation modes

- ▶ Can compile in different modes to get different kinds of programs
- ▶ Can specify these modes very precisely with arguments to `theano.function()`
- ▶ Can use a few quick presets with environment variable flags

Example preset compilation modes

FAST RUN Default. Spends a lot of time on compilation to get an executable that runs fast.

FAST COMPILE Doesn't spend much time compiling. Executable usually uses python instead of compiled C code. Runs slow.

DEBUG MODE Adds lots of checks. Raises error messages in situations other modes don't check for.

Compilation for GPU

- Theano's current back-end only supports 32 bit on GPU
- CUDA supports 64 bit, but is slow in gamer card
- `T.fscalar`, `T.fvector`, `T.fmatrix` are all 32 bit
- `T.scalar`, `T.vector`, `T.matrix` resolve to 32 or 64 bit depending on theano's floatX flag
- floatX is float64 by default, set it to float32
- Set the device flag to gpu (or a specific gpu, like gpu0)

Optimizations

- Theano changes the symbolic expressions you write before converting them to C code
- It makes them faster
  - $(x + y) + (x + y) \rightarrow 2 \times (x + y)$
- It makes them more stable
  - $\exp(a)/\sum \exp(a) \rightarrow \text{softmax}(a)$

Optimizations (2)

Sometimes optimizations discard error checking and produce incorrect output rather than an exception.

```
>>> x = T.scalar()
>>> f = function([x], x/x)
>>> f(0.)
array(1.0)
```

Exercises

Work through the "Compiling and Running" section of the ipython notebook.

Modifying expressions

- The `grad()` method
- Variable nodes
- Types
- Ops
- Apply nodes

The `grad()` method

```
>>> x = T.scalar('x')
>>> y = 2. * x
>>> g = T.grad(y, x)
>>> from theano.printing import min_informative_str
>>> print min_informative_str(g)
A. Elemwise{mul}
 B. Elemwise{second,no_inplace}
  C. Elemwise{mul,no_inplace}
   D. TensorConstant{2.0}
   E. x
  F. TensorConstant{1.0}
 <D>
```

Theano variables

- A *variable* is a theano expression.
- Can come from `T.scalar()`, `T.matrix()`, etc.
- Can come from doing operations on other variables.
- Every variable has a type field, identifying its *type*, such as `TensorType((True, False), 'float32')`
- Variables can be thought of as nodes in a graph

Ops

- ► An Op is any class that describes a function operating on some variables
- ► Can call the op on some variables to get a new variable or variables
- ► An Op class can supply other forms of information about the function, such as its derivative

Apply nodes

- The Apply class is a specific instance of an application of an Op.
- Notable fields:

    op The Op to be applied
    inputs The Variables to be used as input
    outputs The Variables produced

- The owner field on variables identifies the Apply that created it.
- Variable and Apply instances are nodes and owner/ inputs/outputs identify edges in a Theano graph.

Exercises

Work through the "Modifying" section in the ipython notebook.

Debugging

- ▶ DEBUG_MODE
- ▶ Error message
- ▶ `theano.printing.debugprint()`
- ▶ `min_informative_str()`
- ▶ compute_test_value
- ▶ Accessing the FunctionGraph

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 I

```
Traceback (most recent call last):
  File "test.py", line 9, in <module>
    f(np.ones((2,)), np.ones((3,)))
  File "/Users/anakha/Library/Python/2.7/site-packages
     /theano/compile/function_module.py", line 606,
     in __call__
    storage_map=self.fn.storage_map)
  File "/Users/anakha/Library/Python/2.7/site-packages
     /theano/compile/function_module.py", line 595,
     in __call__
    outputs = self.fn()
ValueError: Input dimension mis-match. (input[0].shape
    [0] = 3, input[1].shape[0] = 2)
```

Error message II

```
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.])]
```

Error message III

```
HINT: Re-running with most Theano optimization
    disabled could give you a back-trace of when this
    node was created. This can be done with by setting
    the Theano flag '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 and storage map footprint of this
    apply node.
```

Error message: exception_verbosity=high

```
Debugprint of the apply node :
Elemwise { add , no_inplace } [@A] < TensorType ( float64 ,
    vector ) > ''
 |< TensorType ( float64 , vector ) > [@B] < TensorType (
     float64 , vector ) >
 |< TensorType ( float64 , vector ) > [@C] < TensorType (
     float64 , vector ) >
 |< TensorType ( float64 , vector ) > [@C] < TensorType (
     float64 , vector ) >

Storage map footprint :
 - < TensorType ( float64 , vector ) > , Shape : (3 ,) ,
     ElemSize : 8 Byte ( s ) , TotalSize : 24 Byte ( s )
 - < TensorType ( float64 , vector ) > , Shape : (2 ,) ,
     ElemSize : 8 Byte ( s ) , TotalSize : 16 Byte ( s )
```

Error message: optimizer=fast_compile

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

debugprint

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

min_informative_str

```
>>> x = T.scalar()
>>> y = T.scalar()
>>> z = x + y
>>> z.name = 'z'
>>> a = 2. * z
>>> from theano.printing import min_informative_str
>>> print min_informative_str(a)
A. Elemwise{mul,no_inplace}
 B. TensorConstant{2.0}
 C. z
```

compute test value

```
>>> from theano import config
>>> config.compute_test_value = 'raise'
>>> x = T.vector()
>>> import numpy as np
>>> x.tag.test_value = np.ones((2,))
>>> y = T.vector()
>>> y.tag.test_value = np.ones((3,))
>>> x + y
...
ValueError: Input dimension mis-match.
(input[0].shape[0] = 2, input[1].shape[0] = 3)
```

Accessing a function's fgraph

```
>>> x = T.scalar()
>>> y = x / x
>>> f = function([x], y)
>>> debugprint(f.maker.fgraph.outputs[0])
DeepCopyOp [@A] ''
 |TensorConstant{1.0} [@B]
```

Exercises

Work through the "Debugging" section of the ipython notebook.

Citing Theano

Please cite both of the following papers in all work that uses Theano:

- Bastien, Frédéric, Lamblin, Pascal, Pascanu, Razvan, Bergstra, James, Goodfellow, Ian, Bergeron, Arnaud, Bouchard, Nicolas, and Bengio,Yoshua. Theano: new features and speed improvements. Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop, 2012.

- Bergstra, James, Breuleux, Olivier, Bastien, Frédéric, Lamblin, Pascal, Pascanu, Razvan, Desjardins, Guillaume, Turian, Joseph, Warde- Farley, David, and Bengio,Yoshua. Theano: a CPU and GPU math expression compiler. In Proceedings of the Python for Scientific Computing Conference (SciPy), June 2010. Oral Presentation.

Example acknowledgments

Questions?