Introduction to Theano

Ian Goodfellow
Outline

• 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
Using Theano:

- define expression
- compile expression
- execute expression

\[ f(x, y) = x + y \]
Building expressions

- Scalars
- Vectors
- Matrices
- Tensors
- Reductions
- 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)
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
Exercises

- Work through the “01_building_expressions” directory now
Compiling and running expressions

• theano.function
• shared variables and updates
• compilation modes
• compilation for GPU
• optimizations
```python
>>> 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 function with get_value and set_value
Shared variable example

```python
>>> from theano import shared
>>> x = shared(0.)
>>> from theano.compat.python2x import OrderedDict
>>> updates = OrderedDict()
>>> 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 (thanks, Python!) 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 regard as fine.
Compilation for GPU

- Theano only supports 32 bit on GPU
- CUDA supports 64 bit, but is slow
- \texttt{T.fscalar, T.fvector, T.fmatrix} are all 32 bit
- \texttt{T.scalar, T.vector, T.matrix} resolve to 32 bit or 64 bit depending on theano’s floatX flag
- floatX is float64 by default, set it to float32
- Set 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
  • $(x+y)+(x+y) \rightarrow 2(x + y)$

• It makes them more stable
  • $\exp(a)/\exp(a).\text{sum()} \rightarrow \text{softmax}(a)$
Optimizations

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

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

• Work through the “02_compiling_and_running” directory now
Modifying expressions

• The grad method
• Variable nodes
• Types
• Ops
• Apply nodes
The grad method

```python
>>> 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
  
  • e.g. `TensorType(((True, False), 'float32'))`

• Variables can be thought of as *nodes* in a *graph*
Ops

- An Op is any class that describes a mathematical function of 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 derivatives.
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
  • Variable.owner identifies the Apply that created the variable

• Variable and Apply instances are nodes and owner/inputs/outputs identify edges in a Theano graph
Exercises

• Work through the "03_modifying" directory now
Debugging

- DEBUG_MODE
- compute_test_value
- min_informative_str
- DebugPrint
- Accessing the FunctionGraph
```
>>> 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)
```
>>> 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
```python
>>> 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]
```
Accessing a function's fgraph

```python
>>> 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 “04_debugging” directory now
Citing Theano

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


We would like to thank the developers of Theano—\citep{bergstra+al:2010-scipy,Bastien-Theano-2012}, Pylearn2—\citep{pylearn2_arxiv_2013}. We would also like to thank NSERC, Compute Canada, and Calcul Qu'ebec for providing computational resources.