Theano Tutorials

Ian Goodfellow

Outline

- 1. Basic usage: How to write a theano program
- 2. Advanced usage: Symbolic manipulations and debugging, etc.
- 3A. Internals: how to add your own features
- 3B. Machine learning with Theano

Session 1: Basic Usage

- Overview of Theano (5 min)
- Python basics (20 min)
- Configuring Theano (10 min)
- Building expressions (45 min)
- Compiling and running expressions (45 min)
- Figuring things out (3 min)
- More advanced expressions (45 min)
- Citing Theano (2 min)

Overview of Theano

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

```
int f(int x, int y) {
    return x + y;
}
```

  - execute expression

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

Python basics

- Tuples and lists
- Dictionaries
- Functions and classes
- Exceptions

Tuples and lists

```
>>> l = [1, 2, 3] # make a list
>>> l[1] # index into it
2
>>> l.append(4) # add to it
>>> l
[1, 2, 3, 4]
>>> del l[1] # remove from it
>>> l
[1, 3, 4]
>>> l.insert(1, 3) # insert into it
>>> l
[1, 3, 3, 4]
>>> t = (1, 3, 3, 4) # make a tuple
>>> l == t
False
>>> t[1]
3
>>> del t[1] # tuples are immutable
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object doesn't support item deletion
>>> t2 = tuple(l)
>>> t2
(1, 3, 3, 4)
```

Dictionaries

```
>>> my_dictionary = {}
>>> my_dictionary[1] = 2 # insert a value
>>> my_dictionary[1] # retrieve a value
2
>>> my_dictionary['1'] = 1 # can use any hashable object as a key
>>> my_dictionary[[]] = 1 # lists are not hashable
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unhashable type: 'list'
>>> my_dictionary.keys() # see the contents of a dictionary
['1', 1]
>>> for key in my_dictionary: # iterate over them
...     print key
...
1
1
>>> del my_dictionary[1] # remove an entry
```

Functions and classes

```
>>> def f(x):
...     return 2 * x
...
>>> f(1)
2
>>> class A(object): # Must always inherit from something or you get "old-style class"
...     def __init__(self, p): # This is the constructor
...             self.p = p
...     def f(self, x):
...             return x ** self.p
...
>>> a = A(3)
>>> a.f(2)
8
>>> class B(A):
...     def __init__(self, p):
...             A.__init__(self, p)
...     def f(self, x):
...             return x * p
...
>>> b = B()
>>> b = B(2)
>>> isinstance(b, A)
True
```

Exceptions

```
>>> try:
...     raise ValueError()
... except ValueError:
...     print "Bad value"
...
Bad value
>>> try:
...     raise ValueError()
... except TypeError:
...     print "Bad type"
...
Traceback (most recent call last):
  File "<stdin>", line 2, in <module>
ValueError
```

Configuring Theano

- ~/.theanorc: Settings you always want
- THEANO_FLAGS: Settings for one job
- theano.config: Settings for right now

~/.theanorc

```
[global]
device = cpu
floatX = float32

[warn]
argmax_pushdown_bug = False
sum_div_dimshuffle_bug = False
subtensor_merge_bug = False
```

THEANO_FLAGS

THEANO_FLAGS=”floatX=float64” python my_amazing_theano_script.py

theano.config

```
>>> import theano
>>> theano.config.floatX = 'float32'
>>> x = theano.tensor.scalar()
>>> x.dtype
'float32'
>>> theano.config.floatX = 'float64'
>>> x = theano.tensor.scalar()
>>> x.dtype
'float64'
```

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

```
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

```
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
```

zeros_like and ones_like

- zeros_like(x) returns a symbolic tensor with the same shape and dtype as x, but with every element equal to 0
- ones_like(x) is the same thing, but with 1s

Exercises

- Clone or download the exercises from https://github.com/goodfeli/theano_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

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 function with get_value and set_value

Shared variable example

```
>>> 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
  - T.fscalar, T.fvector, T.fmatrix are all 32 bit
  - 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) -> 2 (x + y)
- It makes them more stable
  - exp(a)/exp(a).sum(axis=1)->softmax(a)

Optimizations

- 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 "02_compiling_and_running" directory now

Figuring things out

- Docstrings:
  - Read the source code
  - help(foo): shows the docstring on foo
- http://deeplearning.net/software/theano/
- theano-users@googlegroups.com

Exercises

- Work through the "03_advanced_expressions" directory now

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.