Tutorial 2
Introduction to Theano

Kai KANG (kkang@ee.cuhk.edu.hk)

Outline

- CUDA/GPU Programming Invited Talks
- Theano Basics
- Beyond Basics
- Theano in Machine Learning and Deep Learning

CUDA/GPU Programming Tutorials

- Speaker: Bin Zhou,PhD
  - NVIDIA CUDA Fellow, USTC Adjunct Research Professor
  - Chief Scientist and Director of Marine Remote Sensing & Information Processing Lab, SDIOI
- First tutorial: 9:30 - 12:15 on Saturday, Jan 24, 2015, LSB LT6
- Second tutorial: 9:30 - 12:15 on Sunday, Jan 25, 2015, TY Wong Hall at SHB

Theano Basics

What is Theano?

- Symbolic computation library
- CPU and GPU infrastructure
- Optimized compiler

Online Materials

- Theano (http://deeplearning.net/software/theano/index.html)
- Theano introduction (http://deeplearning.net/software/theano/introduction.html#introduction)
- Installation guides (http://deeplearning.net/software/theano/install.html#install)
- Theano tutorials (http://deeplearning.net/software/theano/tutorial/index.html#tutorial)
- Documentation (http://deeplearning.net/software/theano/library/index.html#libdoc)
- GitHub page (https://github.com/Theano/Theano)

Algebra Basics

import theano

```
>>> import theano
Using gpu device 0: GeForce GTX 670
>>> import theano.tensor as T
>>> from theano import function
```

define variables

```
>>> x = T.fscalar('x')
>>> y = T.fscalar('y')
>>> x.type
TensorType(float32, scalar)
>>> y.type
TensorType(float32, scalar)
```

Algebra Basics

symbolic operations

```
>>> z = x + y
>>> z.type
TensorType(float32, scalar)
>>> w = x * y
>>> s = z + w
```

pretty print expressions

```
>>> from theano import pp
>>> pp(z)
'(x + y)'
>>> pp(w)
'(x * y)'
>>> s = z + w
>>> pp(s)
'((x + y) + (x * y))'
```

Algebra Basics

compile into functions

```
>>> from theano import function
>>> f1 = function([x, y], z)
>>> f2 = function([x, y], w)
>>> f3 = function([x, y], s)
```

call functions to get NumPy array

```
>>> f1(1.0, 2.0)
array(3.0, dtype=float32)
>>> f2(1.0, 2.0)
array(2.0, dtype=float32)
>>> f3(1.0, 2.0)
array(5.0, dtype=float32)
```

Algebra Basics

- 3 Steps in using Theano
  - Step 1: define symbolic variables

    ```
    >>> x = T.dscalar('x')
    >>> y = T.dscalar('y')
    ```

  - Step 2: define symbolic expressions using defined variables

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

  - Step 3: compile expressions into functions

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

Example

Add two matrices

```
>>> m1 = T.dmatrix()
>>> m2 = T.dmatrix()
>>> sum = m1 + m2
>>> msum = function([m1, m2], sum)

>>> import numpy as np
>>> msum([[1, 3, 5], [7, 9, 11]], [[2, 4, 6], [8, 10, 12]])
array([[  3.,   7.,  11.],
       [ 15.,  19.,  23.]])
>>> msum(np.random.rand(2,3), np.random.rand(2,3))
array([[ 1.07868464,  1.01885801,  0.69778457],
       [ 0.9562832 ,  1.56608957,  1.19840735]])
>>> msum(np.random.rand(2,3), np.random.rand(2,3))
array([[ 1.04984599,  0.42415405,  1.51894434],
       [ 0.80635964,  0.17668743,  1.72312159]])
```

Data Types

- **byte:** bscalar, bvector, bmatrix, brow, bcol, btensor3, btensor4
- **16-bit integers:** wscalar, wvector, wmatrix, wrow, wcol, wtensor3, wtensor4
- **32-bit integers:** iscalar, ivector, imatrix, irow, icol, itensor3, itensor4
- **64-bit integers:** lscalar, lvector, lmatrix, lrow, lcol, ltensor3, ltensor4
- **float:** fscalar, fvector, fmatrix, frow, fcol, ftensor3, ftensor4
- **double:** dscalar, dvector, dmatrix, drow, dcol, dtensor3, dtensor4
- **complex:** cscalar, cvector, cmatrix, crow, ccol, ctensor3, ctensor4

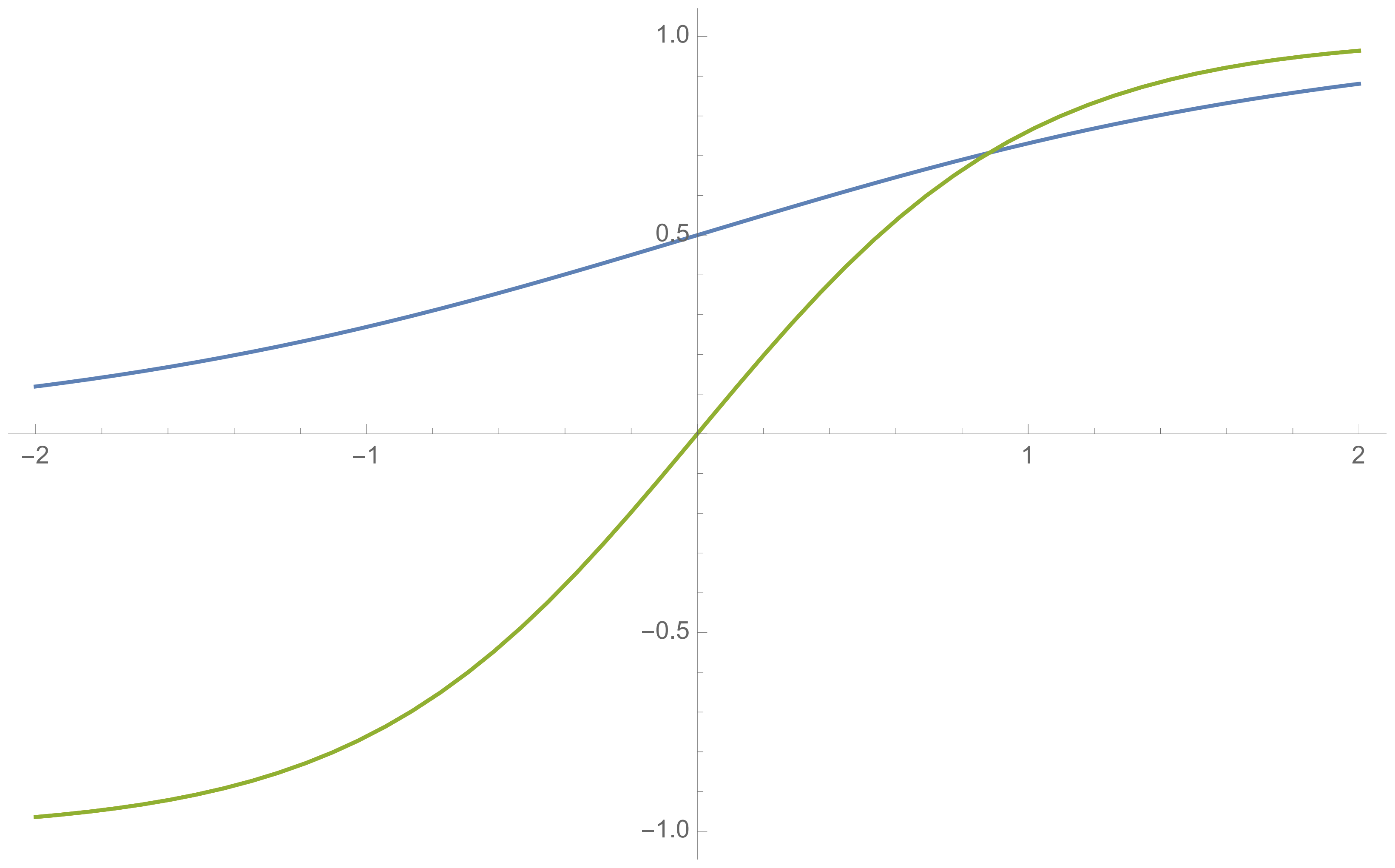
Beyond Basics

Elementwise Operations

```
>>> x = T.dmatrix('x')
>>> s = 1 / (1 + T.exp(-x))
>>> t = T.tanh(x)
>>> sigmoid = function([x], s)
>>> tanh = function([x], t)
>>> sigmoid([[0, 1], [-1, -2]])
array([[ 0.5       ,  0.73105858],
       [ 0.26894142,  0.11920292]])
>>> tanh([[0, 1], [-1, -2]])
array([[ 0.        ,  0.76159416],
       [-0.76159416, -0.96402758]])
>>> 2 * sigmoid([[0, 2], [-2, -4]]) - 1
array([[ 0.        ,  0.76159416],
       [-0.76159416, -0.96402758]])
```

Many mathematical operations, such as exp, log, abs, sin, cos, +, -, *, /, are elementwise operations

Elementwise Operations



```
Plot[{LogisticSigmoid[x],
      Tanh[x],
      2*LogisticSigmoid[2 x]-1},
     {x, -2, 2}]
```

WolframAlpha

Multiple Outputs

Theano functions support multiple outputs in a list

```
>>> a, b = T.dmatrices('a','b')
>>> diff = a - b
>>> abs_diff = abs(diff)
>>> squared_diff = diff ** 2
>>> f = function([a, b], [diff, abs_diff, squared_diff])
>>> f([[1, 1], [1, 1]], [[0, 1], [2, 3]])
[array([[ 1.,  0.],
       [-1., -2.]]),
 array([[ 1.,  0.],
       [ 1.,  2.]]),
 array([[ 1.,  0.],
       [ 1.,  4.]])]
>>> c, d, e = f([[1, 1], [1, 1]], [[0, 1], [2, 3]])
>>> c
array([[ 1.,  0.],
       [-1., -2.]])
>>> d
array([[ 1.,  0.],
       [ 1.,  2.]])
>>> e
array([[ 1.,  0.],
       [ 1.,  4.]])
```

Argument Default Values

Use Param class to specify function parameters

```
>>> from theano import Param
>>> input, step = T.dscalars('input', 'step')
>>> output = input + step
>>> incre = function([input, Param(step, default=1)], output)
>>> incre(33)
array(34.0)
>>> incre(33, 5)
array(38.0)
```

Set parameter values by name

```
>>> m = T.dscalar('m')
>>> output2 = output * m
>>> incre2 = function([input, Param(step, default=1), Param(m,
default=1, name='multiplier')], output2)
>>> incre2(33)
array(34.0)
>>> incre2(33, 5)
array(38.0)
>>> incre2(33, multiplier=2)
array(68.0)
```

Shared Variables

Theano functions can have internal states

```
>>> state = theano.shared(0.)
>>> inc_new = function([Param(step, default=1)], state+step,
updates=[(state, state+step)])
>>> inc_new()
array(1.0)
>>> inc_new()
array(2.0)
>>> inc_new(5)
array(7.0)
>>> inc_new(3)
array(10.0)
>>> inc_new()
array(11.0)
```

'updates' parameter is a list of shared variable and new expression pairs, which set the values of shared variables with those of new expressions

Shared Variables

getters and setters of shared variables

```
>>> state.get_value()
array(11.0)
>>> inc_new()
array(12.0)
>>> state.get_value()
array(12.0)
>>> inc_new(5)
array(17.0)
>>> state.get_value()
array(17.0)
>>> state.set_value(0)
>>> inc_new()
array(1.0)
>>> state.get_value()
array(1.0)
>>> inc_new(5)
array(6.0)
>>> state.get_value()
array(6.0)
```

Shared Variables

Shared variables are shared across functions

```
>>> dec = function([Param(step, default=1)], state-step,
updates=[(state, state-step)])
>>> dec()
array(5.0)
>>> dec(5)
array(0.0)
>>> state.get_value()
array(0.0)
>>> inc_new()
array(1.0)
>>> state.get_value()
array(1.0)
```

Expression Replacement

Use `givens` parameter to replace part of an expression with a new expression

```
>>> x = T.dmatrix('x')
>>> s = 1 / (1 + T.exp(-x))
>>> t = T.tanh(x)
>>> tanh1 = function([x], t)
>>> tanh2 = function([y], 2 * s - 1, givens={x: 2 * y})
>>> tanh1([[0, 1], [3, 5]])
array([[ 0.        ,  0.76159416],
       [ 0.99505475,  0.9999092 ]])
>>> tanh2([[0, 1], [3, 5]])
array([[ 0.        ,  0.76159416],
       [ 0.99505475,  0.9999092 ]])
```

Use theano.clone to do more advanced replacements

```
>>> tanh3 = function([x], theano.clone(2 * s - 1, replace={x: 2 *
x}))
>>> tanh3([[0, 1], [3, 5]])
array([[ 0.        ,  0.76159416],
       [ 0.99505475,  0.9999092 ]])
```

Derivatives

Automatic derivatives with `tensor.grad`

```
>>> x = T.dscalar('x')
>>> y = x ** 2
>>> dy = T.grad(y, wrt=x)
>>> f = function([x], dy)
>>> f(3.0)
array(6.0)
>>> f(5.0)
array(10.0)
```

Theano optimizes expressions when compiling functions

```
>>> pp(dy)
'((fill((x ** TensorConstant{2}), TensorConstant{1.0}) *
TensorConstant{2}) * (x ** (TensorConstant{2} -
TensorConstant{1})))'
>>> pp(f.maker.fgraph.outputs[0])
'(TensorConstant{2.0} * x)'
```

Example - Logistic Regression

```
import numpy
import theano
import theano.tensor as T
rng = numpy.random

N = 400
feats = 784
D = (rng.randn(N, feats), rng.randint(size=N, low=0, high=2))
training_steps = 10000

# Declare Theano symbolic variables
x = T.matrix("x")
y = T.vector("y")
w = theano.shared(rng.randn(feats), name="w")
b = theano.shared(0., name=“b")
print "Initial model:"
print w.get_value(), b.get_value()


# Construct Theano expression graph
p_1 = 1 / (1 + T.exp(-T.dot(x, w) - b))   # Probability that target = 1
prediction = p_1 > 0.5                    # The prediction thresholded
xent = -y * T.log(p_1) - (1-y) * T.log(1-p_1) # Cross-entropy loss function
cost = xent.mean() + 0.01 * (w ** 2).sum()# The cost to minimize
gw, gb = T.grad(cost, [w, b])             # Compute the gradient of the cost
```

Example - Logistic Regression

```
# Compile
train = theano.function(
          inputs=[x,y],
          outputs=[prediction, xent],
          updates=((w, w - 0.1 * gw), (b, b - 0.1 * gb)))
predict = theano.function(inputs=[x], outputs=prediction)

# Train
for i in range(training_steps):
    pred, err = train(D[0], D[1])

print "Final model:"
print w.get_value(), b.get_value()
print "target values for D:", D[1]
print "prediction on D:", predict(D[0])
```

Theano in ML and DL

- Deep learning tutorial based on Theano (http://deeplearning.net/tutorial/contents.html)
    - Getting started tutorial (http://deeplearning.net/tutorial/gettingstarted.html)
- Pylearn2: machine learning library based on Theano (http://deeplearning.net/software/pylearn2/)
- GroundHog: RNN framework based on Theano (https://github.com/lisa-groundhog/GroundHog)
- Faster convolution methods (https://github.com/Theano/Theano/wiki/Convolution-methods)
- …