Practical Deep Neural Networks
GPU computing perspective
Theano Tutorial

Yuhuang Hu       Chu Kiong Loo

Advanced Robotic Lab
Department of Artificial Intelligence
Faculty of Computer Science & IT
University of Malaya
Outline

1. Introduction
2. Baby Steps
3. Complex Examples
4. Derivatives in Theano
5. Conditions and Loop
Outline

1. Introduction
2. Baby Steps
3. Complex Examples
4. Derivatives in Theano
5. Conditions and Loop
Prerequisites

✓ A machine that installed Python and working Theano library.
✓ Know basic Python
✓ Know basic numpy
Suggest Readings

- Python numpy tutorial
  http://cs231n.github.io/python-numpy-tutorial/
- Theano tutorial
  http://deeplearning.net/software/theano/index.html
Outline

1. Introduction
2. Baby Steps
3. Complex Examples
4. Derivatives in Theano
5. Conditions and Loop
import theano.tensor as T
from theano import function

x = T.dscalar('x')
print x.type
y = T.dscalar('y')
z = x + y
f = function([x, y], z)  # The function!

And now that we've created our function we can use it:

print f(2,3);
print f(16.3, 12.1);
Adding two matrices

\[
x = T.dmatrix('x') \\
y = T.dmatrix('y') \\
z = x + y \\
f = function([x, y], z)
\]

```
print f([[1, 2], [3, 4]], [[10, 20], [30, 40]])
```

```
import numpy
print f(numpy.array([[1, 2], [3, 4]]),
    numpy.array([[10, 20], [30, 40]]))
```
Theano is a COMPILER!
Outline

1 Introduction
2 Baby Steps
3 Complex Examples
4 Derivatives in Theano
5 Conditions and Loop
Logistic Function

\[ f(x) = \frac{1}{1 + \exp(-x)} \]

```python
x = T.dmatrix('x')
s = 1 / (1 + T.exp(-x))
logistic = function([x], s)
print logistic([[0, 1], [-1, -2]])
```
Compute more than one thing

```
a, b = T.dmatrices('a', 'b')
diff = a - b
abs_diff = abs(diff)
diff_squared = diff**2
f = function([a, b], [diff, abs_diff, diff_squared])

print f([[1, 1], [1, 1]], [[0, 1], [2, 3]])
```
set default value for argument

```python
from theano import Param
x, y = T.dscalars('x', 'y')
z = x + y
f = function([x, Param(y, default=1)], z)

print f(33)
print f(33, 2)
```
from theano import shared
state = shared(0)
inc = T.iscalar('inc')
accumulator = function([inc], state, updates=[(state, state+inc)])

# get value
print state.get_value()
accumulator(1)
print state.get_value()
accumulator(300)
print state.get_value()

# set value
state.set_value(-1)
decrementedor = function([inc], state, updates=[(state, state-inc)])
decrementedor(2)
print state.get_value()
Using Shared Variable

```python
fn_of_state = state * 2 + inc

# The type of foo must match the shared variable we are replacing
# with the `givens`
foo = T.scalar(dtype=state.dtype)
skip_shared = function([inc, foo], fn_of_state,
                        givens=[(state, foo)])

print skip_shared(1, 3)
print state.get_value()
```
Computing Gradients

```python
from theano import pp
x = T.dscalar('x')
y = x ** 2
gy = T.grad(y, x)
pp(gy)  # print out the gradient prior to optimization
f = function([x], gy)
print f(4)
print f(94.2)
```
Outline

1. Introduction
2. Baby Steps
3. Complex Examples
4. Derivatives in Theano
5. Conditions and Loop
from theano import tensor as T
from theano.ifelse import ifelse
import theano, time, numpy

a,b = T.scalars('a', 'b')
x,y = T.matrices('x', 'y')

z_switch = T.switch(T.lt(a, b), T.mean(x), T.mean(y))
z_lazy = ifelse(T.lt(a, b), T.mean(x), T.mean(y))

f_switch = theano.function([a, b, x, y], z_switch,
                         mode=theano.Mode(linker='vm'))
f_lazyifelse = theano.function([a, b, x, y], z_lazy,
                           mode=theano.Mode(linker='vm'))
import theano
import theano.tensor as T
import numpy as np

# defining the tensor variables
X = T.matrix("X")
W = T.matrix("W")
b_sym = T.vector("b_sym")

results, updates = theano.scan(lambda v: T.tanh(T.dot(v, W) + b_sym),
                               sequences=X)
compute_elementwise = theano.function(inputs=[X, W, b_sym],
                                       outputs=[results])

# test values
x = np.eye(2, dtype=theano.config.floatX)
w = np.ones((2, 2), dtype=theano.config.floatX)
b = np.ones((2), dtype=theano.config.floatX)
b[1] = 2
print compute_elementwise(x, w, b)[0]

# comparison with numpy
print np.tanh(x.dot(w) + b)