LEARNING theano

#theano

Table of Contents

About

Chapter 1: Getting started with theano

Remarks

Theano is a python library, which handles defining and evaluating symbolic expressions over tensor variables. Has various application, but most popular is deep learning.

Examples

Installation or Setup

Detailed instructions on getting theano set up or installed.

Installing Theano and configuring the GPU on Ubuntu 14.04

You can use the following instructions to install Theano and configure the GPU (assume a freshly installed Ubuntu 14.04):

```
# Install Theano
sudo apt-get install python-numpy python-scipy python-dev python-pip python-nose g++
libopenblas-dev git
sudo pip install Theano

# Install Nvidia drivers, CUDA and CUDA toolkit, following some instructions from
http://docs.nvidia.com/cuda/cuda-installation-guide-linux/index.html
wget http://developer.download.nvidia.com/compute/cuda/7.5/Prod/local_installers/cuda-repo-
ubuntu1404-7-5-local_7.5-18_amd64.deb # Got the link at https://developer.nvidia.com/cuda-
downloads
sudo dpkg -i cuda-repo-ubuntu1404-7-5-local_7.5-18_amd64.deb
sudo apt-get update
sudo apt-get install cuda

sudo reboot
```

At that point, running `nvidia-smi` should work, but running `nvcc` won't work.

```
# Execute in console, or (add in ~/.bash_profile then run "source ~/.bash_profile"):
export PATH=/usr/local/cuda-7.5/bin:$PATH
export LD_LIBRARY_PATH=/usr/local/cuda-7.5/lib64:$LD_LIBRARY_PATH
```

At that point, both `nvidia-smi` and `nvcc` should work.

To test whether Theano is able to use the GPU:

Copy-paste the following in `gpu_test.py`:

```
# Start gpu_test.py
# From http://deeplearning.net/software/theano/tutorial/using_gpu.html#using-gpu
from theano import function, config, shared, sandbox
```

```
import theano.tensor as T
import numpy
import time

vlen = 10 * 30 * 768  # 10 x #cores x # threads per core
iters = 1000

rng = numpy.random.RandomState(22)
x = shared(numpy.asarray(rng.rand(vlen), config.floatX))
f = function([], T.exp(x))
print(f.maker.fgraph.toposort())
t0 = time.time()
for i in xrange(iters):
    r = f()
t1 = time.time()
print("Looping %d times took %f seconds" % (iters, t1 - t0))
print("Result is %s" % (r,))
if numpy.any([isinstance(x.op, T.Elemwise) for x in f.maker.fgraph.toposort()]):
    print('Used the cpu')
else:
    print('Used the gpu')
# End gpu_test.py
```

and run it:

```
THEANO_FLAGS='mode=FAST_RUN,device=gpu,floatX=float32' python gpu_test.py
```

which should return:

```
f@f-Aurora-R4:~$ THEANO_FLAGS='mode=FAST_RUN,device=gpu,floatX=float32' python gpu_test.py
Using gpu device 0: GeForce GTX 690
[GpuElemwise{exp,no_inplace}(<CudaNdarrayType(float32, vector)>),
HostFromGpu(GpuElemwise{exp,no_inplace}.0)]
Looping 1000 times took 0.658292 seconds
Result is [ 1.23178029  1.61879349  1.52278066 ...,  2.20771813  2.29967761
  1.62323296]
Used the gpu
```

To know your CUDA version:

```
nvcc -V
```

Example:

```
username@server:~$ nvcc -V
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2015 NVIDIA Corporation
Built on Tue_Aug_11_14:27:32_CDT_2015
Cuda compilation tools, release 7.5, V7.5.17
```

Adding cuDNN

To add cuDNN (instructions from

http://deeplearning.net/software/theano/library/sandbox/cuda/dnn.html):

1. Download cuDNN from https://developer.nvidia.com/rdp/cudnn-download (need registration, which is free)
2. `tar -xvf cudnn-7.0-linux-x64-v3.0-prod.tgz`
3. Do one of the following

Option 1: Copy the `*.h` files to `CUDA_ROOT/include` and the `*.so*` files to `CUDA_ROOT/lib64` (by default, `CUDA_ROOT` is `/usr/local/cuda` on Linux).

```
sudo cp cuda/lib64/* /usr/local/cuda/lib64/
sudo cp cuda/include/cudnn.h /usr/local/cuda/include/
```

Option 2:

```
export LD_LIBRARY_PATH=/home/user/path_to_CUDNN_folder/lib64:$LD_LIBRARY_PATH
export CPATH=/home/user/path_to_CUDNN_folder/include:$CPATH
export LIBRARY_PATH=/home/user/path_to_CUDNN_folder/lib64:$LD_LIBRARY_PATH
```

By default, Theano will detect if it can use cuDNN. If so, it will use it. If not, Theano optimizations will not introduce cuDNN ops. So Theano will still work if the user did not introduce them manually.

To get an error if Theano can not use cuDNN, use this Theano flag: `optimizer_including=cudnn`.

Example:

```
THEANO_FLAGS='mode=FAST_RUN,device=gpu,floatX=float32,optimizer_including=cudnn' python
gpu_test.py
```

To know your cuDNN version:

```
cat /usr/local/cuda/include/cudnn.h | grep CUDNN_MAJOR -A 2
```

Adding CNMeM

The CNMeM library is a "Simple library to help the Deep Learning frameworks manage CUDA memory.".

```
# Build CNMeM without the unit tests
git clone https://github.com/NVIDIA/cnmem.git cnmem
cd cnmem
mkdir build
cd build
sudo apt-get install -y cmake
cmake ..
make

# Copy files to proper location
sudo cp ../include/cnmem.h /usr/local/cuda/include
sudo cp *.so /usr/local/cuda/lib64/
```

```
cd ../..
```

To use with Theano, you need to add the `lib.cnmem` flag. Example:

```
THEANO_FLAGS='mode=FAST_RUN,device=gpu,floatX=float32,lib.cnmem=0.8,optimizer_including=cudnn'
python gpu_test.py
```

The first output of the script should be:

```
Using gpu device 0: GeForce GTX TITAN X (CNMeM is enabled with initial size: 80.0% of memory,
cuDNN 5005)
```

`lib.cnmem=0.8` means that it can use up to 80% of the GPU.

CNMeM has been reported to give some interesting speed improvements, and is supported by Theano, Torch, and Caffee.

Theano - source 1:

> The speed up depend of many factor, like the shapes and the model itself. The speed up go from 0 to 2x faster.

Theano - source 2:

> If you don't change the Theano flag allow_gc, you can expect 20% speed up on the GPU. In some case (small models), we saw a 50% speed up.

Common issues:

- Importing theano: AttributeError: 'module' object has no attribute 'find_graphviz'

Running Theano on multiple CPU cores

You can run Theano on multiple CPU cores with the `OMP_NUM_THREADS=[number_of_cpu_cores]` flag.

Example:

```
OMP_NUM_THREADS=4 python gpu_test.py
```

The script `theano/misc/check_blas.py` outputs information regarding which BLAS is used:

```
cd [theano_git_directory]
OMP_NUM_THREADS=4 python theano/misc/check_blas.py
```

Your first theano program

In this example, we will compile functions that computes sum and difference given two real number.

```
from __future__ import print_function
import theano
import theano.tensor as T

#define two symbolic scalar
s_x = T.fscalar()
s_y = T.fscalar()

#compute something
s_sum = s_x + s_y
s_diff = s_x - s_y

#compile a function that adds two number
#theano will call system compiler at here
fn_add = theano.function(inputs=[s_x, s_y], outputs=s_sum)
fn_diff = theano.function(inputs=[s_x, s_y], outputs=s_diff)

#call the compiled functions
print(fn_add(2., 2.)) #4.
print(fn_diff(2., 2.)) #0.
```

Read Getting started with theano online: https://riptutorial.com/theano/topic/5439/getting-started-with-theano

Chapter 2: Loops with theano

Examples

Basic scan usage

`scan` is used for calling function multiple times over a list of values, the function may contain state.

`scan` syntax (as of theano 0.9):

```
scan(
    fn,
    sequences=None,
    outputs_info=None,
    non_sequences=None,
    n_steps=None,
    truncate_gradient=-1,
    go_backwards=False,
    mode=None,
    name=None,
    profile=False,
    allow_gc=None,
    strict=False)
```

This can be very confusing at a first glance. We will explain several basic but important `scan` usage in multiple code examples.

The following code examples assume you have executed imports:

```
import numpy as np
import theano
import theano.tensor as T
```

`sequences` - Map a function over a list

In the simplest case, scan just maps a pure function (a function without state) to a list. The lists is specified in the `sequences` argument

```
  s_x = T.ivector()
  s_y, _ = theano.scan(
      fn = lambda x:x*x,
      sequences = [s_x])
  fn = theano.function([s_x], s_y)
  fn([1,2,3,4,5]) #[1,4,9,16,25]
```

Note `scan` have two return values, the former is the resulting list, and the latter is the updates to state value, which will be explained later.

`sequences` - Zip a function over a list

Almost same as above, just give `sequences` argument a list of two elements. The order of the two elements should match to the order of arguments in `fn`

```
s_x1 = T.ivector()
s_x2 = T.ivector()
s_y, _ = theano.scan(
    fn = lambda x1,x2:x1**x2,
    sequences = [s_x1, s_x2])
fn = theano.function([s_x], s_y)
fn([1,2,3,4,5],[0,1,2,3,4]) #[1,2,9,64,625]
```

`outputs_info` - Accumulate a list

Accumulation involves a state variable. State variables need initial values, which shall be specified in the `outputs_info` parameter.

```
s_x = T.ivector()
v_sum = th.shared(np.int32(0))
s_y, update_sum = theano.scan(
    lambda x,y:x+y,
    sequences = [s_x],
    outputs_info = [s_sum])
fn = theano.function([s_x], s_y, updates=update_sum)

v_sum.get_value() # 0
fn([1,2,3,4,5]) # [1,3,6,10,15]
v_sum.get_value() # 15
fn([-1,-2,-3,-4,-5]) # [14,12,9,5,0]
v_sum.get_value() # 0
```

We put a shared variable into `outputs_info`, this will cause `scan` return updates to our shared variable, which can then be put into `theano.function`.

`non_sequences` and `n_steps` - Orbit of logistic map `x -> lambda*x*(1-x)`

You can give inputs that does not change during `scan` in `non_sequences` argument. In this case `s_lambda` is a non-changing variable (but NOT a constant since it must be supplied during runtime).

```
s_x = T.fscalar()
s_lambda = T.fscalar()
s_t = T.iscalar()
s_y, _ = theano.scan(
    fn = lambda x,l: l*x*(1-x),
    outputs_info = [s_x],
    non_sequences = [s_lambda],
    n_steps = s_t
)
fn = theano.function([s_x, s_lambda, s_t], s_y)

fn(.75, 4., 10) #a stable orbit

#[ 0.75,  0.75,  0.75,  0.75,  0.75,  0.75,  0.75,  0.75,  0.75,  0.75]

fn(.65, 4., 10) #a chaotic orbit

#[ 0.91000003,  0.32759991,  0.88111287,  0.41901192,  0.97376364,
  # 0.10219204,  0.3669953 ,  0.92923898,  0.2630156 ,  0.77535355]
```

Taps - Fibonacci

states/inputs may come in multiple timesteps. This is done by:

- putting `dict(input=<init_value>, taps=<list of int>)` inside `sequences` argument.
- putting `dict(initial=<init_value>, taps=<list of int>)` inside `outputs_info` argument.

In this example, we use two taps in `outputs_info` to compute recurrence relation `x_n = x_{n-1} + x_{n-2}`.

```
s_x0 = T.iscalar()
s_x1 = T.iscalar()
s_n = T.iscalar()
s_y, _ = theano.scan(
    fn = lambda x1,x2: x1+x2,
    outputs_info = [dict(initial=T.join(0,[s_x0, s_x1]), taps=[-2,-1])],
    n_steps = s_n
)
fn_fib = theano.function([s_x0, s_x1, s_n], s_y)
fn_fib(1,1,10)
# [2, 3, 5, 8, 13, 21, 34, 55, 89, 144]
```

theano map and reduce

`theano.map` and `theano.scan_module.reduce` are wrappers of `theano_scan`. They can be seen as handicapped version of `scan`. You can view **Basic scan usage** section for reference.

```
import theano
import theano.tensor as T
s_x = T.ivector()
s_sqr, _ = theano.map(
    fn = lambda x:x*x,
    sequences = [s_x])
s_sum, _ = theano.reduce(
    fn = lambda: x,y:x+y,
    sequences = [s_x],
    outputs_info = [0])
fn = theano.function([s_x], [s_sqr, s_sum])
fn([1,2,3,4,5]) #[1,4,9,16,25], 15
```

making while loop

As of theano 0.9, while loops can be done via `theano.scan_module.scan_utils.until`. To use, you should return `until` object in `fn` of `scan`.

In the following example, we build a function that checks whether a complex number is inside Mandelbrot set. A complex number `z_0` is inside mandelbrot set if series `z_{n+1} = z_{n}^2 + z_0` does not converge.

```
MAX_ITER = 256
BAILOUT = 2.
s_z0 = th.cscalar()
def iterate(s_i_, s_z_, s_z0_):
    return [s_z_*s_z_+s_z0_,s_i_+1], {}, until(T.abs_(s_z_)>BAILOUT)
(_1, s_niter), _2 = theano.scan(
    fn = iterate,
    outputs_info = [0, s_z0],
    non_sequences = [s_z0],
    n_steps = MAX_ITER
)
fn_mandelbrot_iters = theano.function([s_z0], s_niter)
def is_in_mandelbrot(z_):
    return fn_mandelbrot_iters(z_)>=MAX_ITER

is_in_mandelbrot(0.24+0.j) # True
is_in_mandelbrot(1.j) # True
is_in_mandelbrot(0.26+0.j) # False
```

Read Loops with theano online: https://riptutorial.com/theano/topic/7609/loops-with-theano

Credits

S. No	Chapters	Contributors
1	Getting started with theano	Alleo, Community, Franck Dernoncourt, Kh40tiK
2	Loops with theano	Kh40tiK