

Instructor-Led Lab: Image Classification using the Theano Python Library

Frédéric Bastien

Montreal Institute for Learning Algorithms
Université de Montréal
Montréal, Canada
`bastienf@iro.umontreal.ca`

Presentation prepared with Pierre Luc Carrier and Arnaud Bergeron



GTC 2017

Université 
de Montréal

Slides

- ▶ github repo of this presentation
<https://github.com/nouiz/gtc2017/>

Introduction

Community

Theano

Compiling/Running

Modifying expressions

GPU

Debugging

Models

Logistic Regression

Convolution

Lasagne

Exercises

End

High level

Python <- {NumPy/SciPy/libgpuarray} <- Theano <- {...}

- ▶ Python: OO coding language
- ▶ Numpy: n -dimensional array object and scientific computing toolbox
- ▶ SciPy: sparse matrix objects and more scientific computing functionality
- ▶ libgpuarray: GPU n -dimensional array object in C for CUDA and OpenCL(not ready!)
- ▶ Theano: compiler/symbolic graph manipulation
 - ▶ **(Not a machine learning framework/software)**
- ▶ {...}: Many libraries built on top of Theano

What Theano provides

- ▶ Lazy evaluation for performance
- ▶ GPU support
- ▶ Symbolic differentiation
- ▶ Automatic speed and stability optimization

High level

Many [machine learning] library build on top of Theano

- ▶ Keras
- ▶ lasagne
- ▶ PyMC 3
- ▶ blocks
- ▶ sklearn-theano
- ▶ theano-rnn
- ▶ Morb
- ▶ ...

Goal of the stack

Fast to develop
Fast to run



Some models build with Theano

Some models that have been build with Theano.

- ▶ Neural Networks
- ▶ Convolutional NN: CNN, AlexNet, OverFeat, GoogLeNet, Inception, ...
- ▶ Recurrent NN: RNN, CTC, LSTM, GRU, attention mechanisms, ...
- ▶ NADE, RNADE, MADE
- ▶ Autoencoders: AE, VAE, ...
- ▶ Generative Adversarial Nets
- ▶ SVMs
- ▶ **many variations of above models and more**

Project status

- ▶ Mature: Theano has been developed and used since January 2008 (9 yrs old)
- ▶ Driven hundreds of research papers
- ▶ Good user documentation
- ▶ Active mailing list with worldwide participants
- ▶ Core technology for Silicon-Valley start-ups
- ▶ Many contributors (some from outside our institute)
- ▶ Used to teach many university classes
- ▶ Used for research at big companies
- ▶ Theano 0.9 released 20th of March, 2017

Theano: deeplearning.net/software/theano/

Deep Learning Tutorials: deeplearning.net/tutorial/

Theano community

Active community

- ▶ Many people reply on our mailing lists
- ▶ Hundreds of answered questions on StackOverflow
- ▶ 123 contributors to Theano 0.9
- ▶ Main developers at MILA

Python

- ▶ General-purpose high-level OO interpreted language
- ▶ Emphasizes code readability
- ▶ Comprehensive standard library
- ▶ Dynamic type and memory management
- ▶ Easily extensible with C
- ▶ Slow execution
- ▶ Popular in *web development* and *scientific communities*

NumPy/SciPy

- ▶ NumPy provides an n -dimensional numeric array in Python
 - ▶ Perfect for high-performance computing
 - ▶ Slices of arrays are views (no copying)
- ▶ NumPy provides
 - ▶ Elementwise computations
 - ▶ Linear algebra, Fourier transforms
 - ▶ Pseudorandom number generators (many distributions)
- ▶ SciPy provides lots more, including
 - ▶ Sparse matrices
 - ▶ More linear algebra
 - ▶ Solvers and optimization algorithms
 - ▶ Matlab-compatible I/O
 - ▶ I/O and signal processing for images and audio

Introduction

Community

Theano

Compiling/Running

Modifying expressions

GPU

Debugging

Models

Logistic Regression

Convolution

Lasagne

Exercises

End

Description

High-level domain-specific language for numeric computation.

- ▶ **Syntax as close to NumPy as possible**
- ▶ **Compiles** most common expressions to **C** for **CPU** and/or **GPU**
- ▶ **Limited expressivity** means more opportunities for optimizations
 - ▶ **Strongly typed** -> compiles to C
 - ▶ **Array oriented** -> easy parallelism
 - ▶ Support for **looping and branching** in expressions
 - ▶ No subroutines -> **global optimization**
- ▶ **Automatic** speed and numerical stability **optimizations**

Description (2)

- ▶ **Symbolic differentiation and R op** (Hessian Free Optimization)
- ▶ Can **reuse other technologies** for best performance
 - ▶ CUDA, CuBLAS, CuDNN, BLAS, SciPy, PyCUDA, Cython, Numba, ...
- ▶ Works on **Linux, OS X and Windows**
- ▶ **Multi-GPU** (via platoon)
- ▶ New GPU back-end:
 - ▶ **Float16 storage** new back-end (need cuda 7.5)
 - ▶ **Multi dtypes**
 - ▶ **Much simpler installation on Windows**
- ▶ Extensive unit-testing and self-verification
- ▶ Extensible (You can create new operations as needed)

Simple example

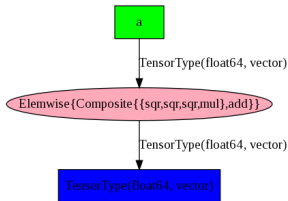
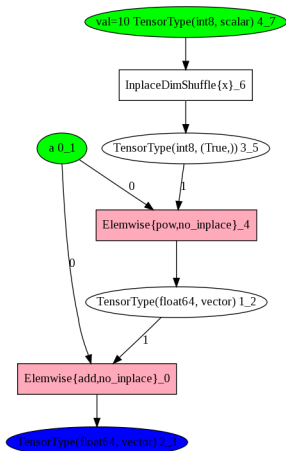
```
import theano
# declare symbolic variable
a = theano.tensor.vector("a")

# build symbolic expression
b = a + a ** 10

# compile function
f = theano.function([a], b)

# Execute with numerical value
print f([0, 1, 2])
# prints 'array([0, 2, 1026])'
```


Simple example



Overview of library

Theano is many things

- ▶ Language
- ▶ Compiler
- ▶ Python library

Scalar math

Some example of scalar operations:

```
import theano
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 (as NumPy, very powerful)
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

Using Theano:

- ▶ Dimensionality defined by length of “broadcastable” argument
- ▶ Can add (or do other elemwise op) 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 = T.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))
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
```


Indexing

As NumPy! This mean slices and index selection return view

return views, supported on GPU

`a_tensor[int]`

`a_tensor[int, int]`

`a_tensor[start:stop:step, start:stop:step]`

`a_tensor[::-1]` *# reverse the first dimension*

Advanced indexing, return copy

`a_tensor[an_index_vector]` *# Supported on GPU*

`a_tensor[an_index_vector, an_index_vector]`

`a_tensor[int, an_index_vector]`

`a_tensor[an_index_tensor, ...]`

Compiling and running expression

- ▶ `theano.function`
- ▶ shared variables and updates
- ▶ compilation modes

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
- ▶ Can modify value outside of Theano function with `get_value()` and `set_value()` methods.

Shared variable example

```
>>> from theano import shared
>>> x = shared(0.)
>>> updates = [(x, x + 1)]
>>> f = function([], updates=updates)
>>> f()
>>> x.get_value()
1.0
>>> x.set_value(100.)
>>> f()
>>> x.get_value()
101.0
```

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

Interesting compilation configuration

Some Theano flags:

- ▶ `mode=FAST_RUN`: default. Fastest execution, slowest compilation
- ▶ `mode=FAST_COMPILE`: Fastest compilation, slowest execution. No C code.
- ▶ `mode=DEBUG_MODE`: Adds lots of checks.
- ▶ `optimizer=fast_compile`: `mode=FAST_COMPILE` with C code.
- ▶ `optimizer=stabilize`: `optimizer=fast_compile` with stability optimization.

Theano flags

Can be set globally:

- ▶ In a configuration file `~/.theanorc`
- ▶ `THEANO_FLAGS=mode=FAST_COMPILE python script.py`

Sometimes as parameter of functions:

- ▶ `theano.function(..., mode="FAST_COMPILE")`

Modifying expressions

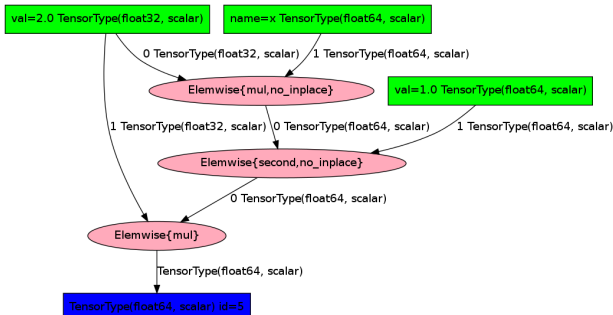
There are “macro” that automatically build bigger graph for you.

- ▶ theano.grad
- ▶ Others

Those functions can get called many times, for example to get the 2nd derivative.

The grad method

```
>>> x = T.scalar('x')
>>> y = 2. * x
>>> g = T.grad(y, x)
# Print the not optimized graph
>>> theano.printing.pydotprint(g)
```

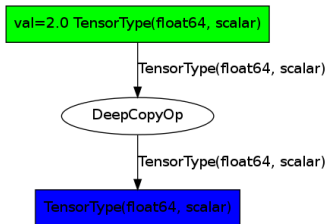


The grad method

```
>>> x = T.scalar('x')  
>>> y = 2. * x  
>>> g = T.grad(y, x)
```

Print the optimized graph

```
>>> f = theano.function([x], g)  
>>> theano.printing.pydotprint(f)
```



Others

- ▶ R_op, L_op for Hessian Free Optimization
- ▶ hessian
- ▶ jacobian
- ▶ clone the graph with replacement
- ▶ you can navigate the graph if you need (go from the result of computation to its input, recursively)

Enabling GPU

- ▶ `libgpuarray` (new-backend) supports all dtype
 - ▶ including `float16` for storage
- ▶ Theano's old GPU back-end removed from the master of Theano
- ▶ CUDA supports `float64`, but it is slow on gamer GPUs

CuDNN

- ▶ V5 and V5.1 are supported
- ▶ V6 compile
- ▶ It is enabled automatically if available
- ▶ Theano flag to get an error if can't be used:
“dnn.enabled=True”
- ▶ Theano flag to disable it: “dnn.enabled=False”

GPU: Theano flags

Theano flags allow to configure Theano. Can be set via a configuration file or an environment variable.

To enable GPU:

- ▶ Set “device=cuda” (or a specific GPU, like “cuda0”)
- ▶ Set “floatX=float32”
- ▶ Optional: warn_float64={'ignore', 'warn', 'raise', 'pdb'}
- ▶ Instead of Theano flags, user can call
“theano.gpuarray.use('cuda0')”

floatX

Allow to change the dtype between float32 and float64.

- ▶ T.fscalar, T.fvector, T.fmatrix are all 32 bit
- ▶ T.dscalar, T.dvector, T.dmatrix are all 64 bit
- ▶ T.scalar, T.vector, T.matrix resolve to floatX
- ▶ floatX is float64 by default, set it to float32 for GPU

Debugging

- ▶ DebugMode: a mode that tests many things done by Theano (very slow)
- ▶ NanGuardMode: a mode that help find the cause of nan in the graph
- ▶ Error message
- ▶ theano.printing.debugprint: print a textual representation of computation
- ▶ profiling: To help know where time is spend

Error message: code

```
import numpy as np
import theano
import theano.tensor as T
x = T.vector()
y = T.vector()
z = x + x
z = z + y
f = theano.function([x, y], z)
f(np.ones((2,)), np.ones((3,)))
```

Error message: 1st part

```
Traceback (most recent call last):
[...]
ValueError: Input dimension mis-match.
    (input[0].shape[0] = 3, input[1].shape[0] = 2)
Apply node that caused the error:
    Elemwise{add,no_inplace}(<TensorType(float64, vector)>,
                             <TensorType(float64, vector)>,
                             <TensorType(float64, vector)>)
Inputs types: [TensorType(float64, vector),
               TensorType(float64, vector),
               TensorType(float64, vector)]
Inputs shapes: [(3,), (2,), (2,)]
Inputs strides: [(8,), (8,), (8,)]
Inputs values: [array([ 1.,  1.,  1.]),
               array([ 1.,  1.]),
               array([ 1.,  1.])]
Outputs clients: [['output']]
```

Error message: 2st part

HINT: Re-running with most Theano optimization disabled could give you a back-traces when this node was created. This can be done with by setting the Theano flags “optimizer=fast_compile”. If that does not work, Theano optimizations can be disabled with “optimizer=None”.

HINT: Use the Theano flag “exception_verbosity=high” for a debugprint of this apply node.

Error message: traceback

Traceback (most recent call last):

```
File "test.py", line 9, in <module>
```

```
    f(np.ones((2,)), np.ones((3,)))
```

```
File "/u/bastienf/repos/theano/compile/function_module.py",
```

```
    line 589, in __call__
```

```
        self.fn.thunks[self.fn.position_of_error])
```

```
File "/u/bastienf/repos/theano/compile/function_module.py",
```

```
    line 579, in __call__
```

```
    outputs = self.fn()
```

Error message: optimizer=fast_compile

Backtrace when the node is created:
File "test.py", line 7, in <module>
 z = z + y

debugprint

```
>>> from theano.printing import debugprint
>>> debugprint(a)
Elemwise{mul,no_inplace} [id A]  ' '
| TensorConstant{2.0} [id B]
| Elemwise{add,no_inplace} [id C]  'z'
| <TensorType(float64, scalar)> [id D]
| <TensorType(float64, scalar)> [id E]
```

Introduction

Community

Theano

Compiling/Running

Modifying expressions

GPU

Debugging

Models

Logistic Regression

Convolution

Lasagne

Exercises

End

Inputs

```
# Load from disk and put in shared variable.  
datasets = load_data(dataset)  
train_set_x, train_set_y = datasets[0]  
valid_set_x, valid_set_y = datasets[1]
```

```
# allocate symbolic variables for the data  
index = T.iscalar() # index to a [mini]batch
```

```
# generate symbolic variables for input minibatch  
x = T.matrix('x') # data, 1 row per image  
y = T.ivector('y') # labels
```

Model

```
n_in = 28 * 28  
n_out = 10
```

```
# weights
```

```
W = theano.shared(  
    numpy.zeros((n_in, n_out),  
                dtype=theano.config.floatX))
```

```
# bias
```

```
b = theano.shared(  
    numpy.zeros((n_out, ),  
                dtype=theano.config.floatX))
```

Computation

the forward pass

```
p_y_given_x = T.nnet.softmax(T.dot(input, W) + b)
```

cost we minimize: the negative log likelihood

```
l = T.log(p_y_given_x)
```

```
cost = -T.mean(l[T.arange(y.shape[0]), y])
```

the error

```
y_pred = T.argmax(p_y_given_x, axis=1)
```

```
err = T.mean(T.neq(y_pred, y))
```

Gradient and updates

compute the gradient of cost

```
g_W, g_b = T.grad(cost=cost, wrt=(W, b))
```

model parameters updates rules

```
updates = [(W, W - learning_rate * g_W),  
            (b, b - learning_rate * g_b)]
```

Training function

```
# compile a Theano function that train the model
train_model = theano.function(
    inputs=[index], outputs=(cost, err),
    updates=updates,
    givens={
        x: train_set_x[index * batch_size:
                        (index + 1) * batch_size],
        y: train_set_y[index * batch_size:
                        (index + 1) * batch_size]
    }
)
```

Introduction

Community

Theano

Compiling/Running

Modifying expressions

GPU

Debugging

Models

Logistic Regression

Convolution

Lasagne

Exercises

End

Inputs

```
# Load from disk and put in shared variable.
```

```
datasets = load_data(dataset)
```

```
train_set_x, train_set_y = datasets[0]
```

```
valid_set_x, valid_set_y = datasets[1]
```

```
# allocate symbolic variables for the data
```

```
index = T.lscalar() # index to a [mini]batch
```

```
x = T.matrix('x') # the data, 1 row per image
```

```
y = T.ivector('y') # labels
```

```
# Reshape matrix of shape (batch_size, 28 * 28)
```

```
# to a 4D tensor, compatible for convolution
```

```
layer0_input = x.reshape((batch_size, 1, 28, 28))
```

Model

```
image_shape=(batch_size, 1, 28, 28)
filter_shape=(nkerns[0], 1, 5, 5)

W_bound = ...
W = theano.shared(
    numpy.asarray(
        rng.uniform(low=-W_bound, high=W_bound,
                    size=filter_shape),
        dtype=theano.config.floatX))

# the bias is a 1D tensor
# one bias per output feature map
b_values = numpy.zeros((filter_shape[0],), dtype=...)
b = theano.shared(b_values)
```


Computation

```
# convolve input feature maps with filters
conv_out = nnet.conv2d(input=x, filters=W)

# pool each feature map individually ,
# using maxpooling
pooled_out = pool.pool_2d(
    input=conv_out,
    ds=(2, 2), // poolsize
    ignore_border=True)

output = T.tanh(pooled_out +
                 b.dimshuffle('x', 0, 'x', 'x'))
```

Introduction

Community

Theano

Compiling/Running

Modifying expressions

GPU

Debugging

Models

Logistic Regression

Convolution

Lasagne

Exercises

End

What is Lasagne

Lasagne is a thin framework/library on top of Theano.

<http://lasagne.readthedocs.org/>

- ▶ Does not hide Theano
- ▶ Easily build Theano graphs by using layers
- ▶ Contains many preimplemented losses and optimizers
- ▶ Does not include a training loop

Lasagne MLP Example: Input Variables

```
input_var = T.tensor4('inputs')  
target_var = T.ivector('targets')
```

Lasagne MLP Example: Model

Input and dropout layer

```
net = lasagne.layers.InputLayer(  
    shape=(None, 1, 28, 28), input_var=input_var)  
net = lasagne.layers.DropoutLayer(net, p=0.2)
```

Hidden layers and dropout:

```
nonlin = lasagne.nonlinearities.rectify  
for _ in range(2):  
    net = lasagne.layers.DenseLayer(  
        network, 800, nonlinearity=nonlin)  
    net = lasagne.layers.dropout(network, p=0.5)
```

Output layer:

```
softmax = lasagne.nonlinearities.softmax  
net = lasagne.layers.DenseLayer(network, 10,  
    nonlinearity=softmax)
```

Lasagne MLP Example: Train Function

```

pred = lasagne.layers.get_output(network)
cat_cross_ent = lasagne.objectives.
    categorical_crossentropy
loss = cat_cross_ent(pred, target_var)
loss = loss.mean()

params = lasagne.layers.get_all_params(
    network, trainable=True)
updates = lasagne.updates.nesterov_momentum(
    loss, params, learning_rate=0.01,
    momentum=0.9)

train_fn = theano.function([input_var, target_var],
                             loss, updates=updates)
  
```

Lasagne MLP Example: Test Function

```
test_pred = lasagne.layers.get_output(  
    network, deterministic=True)  
test_loss = cat_cross_ent(test_pred, target_var)  
test_loss = test_loss.mean()  
  
test_acc = T.mean(T.eq(T.argmax(test_pred, axis=1),  
                        target_var),  
                  dtype=theano.config.floatX)  
  
val_fn = theano.function([input_var, target_var],  
                          [test_loss, test_acc])
```

Introduction

Community

Theano

Compiling/Running

Modifying expressions

GPU

Debugging

Models

Logistic Regression

Convolution

Lasagne

Exercises

End

ipython notebook

- ▶ Introduction
- ▶ Exercises (Theano only exercises)
- ▶ lenet (small CNN model to quickly try it)
- ▶ Reuse VGG16 features

Connection instructions

- ▶ Navigate to `nvmlabs.qwiklab.com`
- ▶ Login or create a new account
- ▶ Select the “Instructor-Led Hands-on Labs” class
- ▶ Find the lab called “Theano” and click Start
- ▶ After a short wait, lab instance connection information will be shown
- ▶ Please ask Lab Assistants for help!

Further hands-on training

Check out the Self-Paced labs at the conference:

- ▶ Deep Learning, CUDA, OpenACC, Tools and more!
- ▶ Just grab a seat and take any available lab
- ▶ Located in the lower-level outside of LL20C

You will also receive Credits to take additional labs at nvidia.qwiklab.com

- ▶ Log in using the same account you used in this lab

Where to learn more

- ▶ Deep Learning Tutorials with Theano:
deeplearning.net/tutorial
- ▶ Theano tutorial: deeplearning.net/software/tutorial
- ▶ Theano website: deeplearning.net/software
- ▶ Lasagne documentation: <http://lasagne.readthedocs.io/>
- ▶ You can also see frameworks on top of Theano like Blocks, Keras, Lasagne, ...

Questions, acknowledgments

Questions? Acknowledgments

- ▶ All people working or having worked at the MILA institute
- ▶ All Theano users/contributors
- ▶ Compute Canada, RQCHP, NSERC, NVIDIA, and Canada Research Chairs for providing funds, access to computing resources, hardware or GPU libraries.