Theano Tutorials
Session 3
Internals

Ian Goodfellow

Outline

- Social structure of theano development
- Writing a new Op
- Writing an optimization
- Writing GPU ops
- Features we're interested in

Social structure

- Mailing lists
- Development team
- Using github

Mailing lists

- theano-users@googlegroups.com
  - Write here if you are having a problem understanding how to use provided Theano classes and functions
  - Bug reports OK here
- theano-dev@googlegroups.com
  - Bug reports more likely to get prompt attention here
  - Ask for help writing new Theano Ops, optimizations, etc.
  - Propose / ask for help designing new Theano features

Development team

- Frédéric Bastien is a software engineer on staff at LISA lab. He is the main person presently maintaining / developing Theano
- LISA students participate in a "Common Code Workflow" (a kind of incorrect translation from French name, should be more of "Shared Library Development") program
- The CCW sometimes adds Theano features buts mostly works on Pylearn2
- I myself am a fairly minor contributor; I've mostly done speedups, bug fixes, better error messages, made behavior more consistent, etc. but few new features

Using GitHub

- Make an account on github
- Use the "fork" feature to make your own Theano repository in your account
- "git clone" the repository to your machine
- "git checkout -b my_feature master" to start working in a branch to develop your new feature
- "git add" and "git commit" to log your changes in your branch
- "git push origin my_feature" to send the branch to your fork on your GitHub account
- Use the "pull request" feature from your repository's GitHub page to ask the Theano developers to merge your branch into the "master" branch of the central repository
- Theano developers will comment on your code and request changes
- Make the changes, use "git add" and "git commit" to track then, and "git push origin my_feature" to add them to the pull request

Writing a new Op

- The Op contract
- Op.grad
- How theano.grad works
- verify_grad
- Op.perform
- Op.c_code

The Op contract

- There is no Op base class to inherit from
- Instead, write any class that obeys "the Op contract"
- Described here: http://deeplearning.net/software/theano/extending/op.html#op-s-contract

Op.grad

- Builds an expression for the product between the op's Jacobian and another vector
- theano.grad will pass in the gradient of the cost on the outputs as that vector
- Described in detail here: http://deeplearning.net/software/theano/extending/op.html#grad

How theano.grad works

- https://github.com/Theano/Theano/blob/master/theano/gradient.py#L354

verify_grad

- Uses numerical differentiation to make sure your symbolic differentiation is accurate
- Numerical differentiation can be expensive for multiple inputs / outputs. Compensate by random linear projection from one input to many inputs, many outputs to one output.

Op.perform

- Python code for computing the op
- Many operations are slow when implemented in python, but this can provide a good reference implementation for DEBUG_MODE
- Some operations can be implemented efficiently by calling a few other object's C bindings from python code

Op.c_code

- Returns a string for C code to carry out the op's calculations
- Change the op's c_code_cache_version each time you change this, otherwise it won't get regenerated

Writing an optimization

```
#numerically stabilize log softmax (X)
# as  X-X.max(axis=1).dimshuffle(0,'x') - log(exp(X-X.max(axis=1).dimshuffle(0,'x')).sum(axis=1)).dimshuffle(0,'x)
def make_out_pattern(X):
    stabilized_X = X - X.max(axis=1).dimshuffle(0, 'x')
    out_var = stabilized_X - tensor.log(tensor.exp(stabilized_X).sum(
        axis=1)).dimshuffle(0, 'x')
    # tell DEBUG_MODE that it's OK if the original graph produced NaN and the optimized graph does not
    out_var.values_eq_approx = out_var.type.values_eq_approx_remove_nan
    return out_var


local_log_softmax = gof.PatternSub(in_pattern=(tensor.log, (softmax, 'x')),
                                   out_pattern=(make_out_pattern, 'x'),
                                  allow_multiple_clients=True)

# Register the optimization
opt.register_specialize(local_log_softmax, name='local_log_softmax')
```

Views and inplace optimizations

- Views
  - Variables whose representation depends on another variable (usually to save memory)
  - Example: y = x[i,:]
- Inplace operations
  - Example: z = T.nnet.sigmoid(x)
  - To save memory, we may want to compute z in the same buffer that was used to store x
- But what if someone is still using y?

The DestroyHandler

- User is not allowed to manually insert inplace ops
- User builds a graph with non-inplace ops
- Optimizations propose turning ops inplace
- Destroyhandler evaluates graph for cyclical dependencies
  - If a cycle exists no creation/deletion schedule is possible and the optimization is rejected

Writing a GPU op

- Write a CPU version (the actual implementation is optional)
- Write a GPU version
- Write an optimization to turn the CPU version into a GPU version
- User allocates CPU version
- Optimizations turn it into the GPU version when running with device=gpu

Features we're interested in

- Built-in mode for disabling graph optimizations
- Improved shared variables
- Reduced python dependency

Graph optimization disabling mode

- A convenient thing to do is run with all ops using C code but not doing any graph optimizations (basically, do what Torch always does)
- Currently you can do this by manually constructing the right mode in your python code
- It would be nice to have a C_CODE_MODE default
- Some rough edges on the GPU

Improved shared variables

- Shared variables are Theano's only way of modifying a buffer
- Currently each Theano function is compiled in terms of specific variables, e.g. buffers
- Could be nice to run the same function on many buffers
- GPU serialization issue

Reduced python dependency

- Python has a Global Interpreter Lock
- GIL means that multiple threads can't accomplish much CPU parallelism
- Much of the C code depends on numpy libraries, and thus python memory management