GSoC 2016 Application

Sub org info: Theano

Student Information:

- Name: S. Ramana Subramanyam
- Alternative name: sentient07
- Email: vxrram95@gmail.com
- Telephone: +91 9840340649
- Time zone: UTC + 5:30
- Chat availability : 10AM to 4:30PM EDT on all weekdays
- Blog (Exclusively for GSoC) - http://sentient07.blogspot.in

Code Samples:

- I have been quite an active contributor to *theano* from the end November. Here are some of my Pull Requests, sorted by date (at the time of writing)
  - PR#3673 - Fixes issue#3190
  - PR#3697 - Fixes issue#3687 : Wrote test for `DownsampleFactorMax`'s output shape.
  - PR#3747 - Fixes issue#3094 : Made make_node optional when itypes and otypes are defined while inheriting from a theano OP
  - PR#3284-Fixes issue#3686 : Added test for `GetConvOutShape`'s method.
  - PR#3999- Better adaptability of Theano with Python3. Changed the behaviour of theano with Python3, in sync with numpy.
  - PR#4013 - fixes issue#3573 : Made ApplyNode reintroduce into the fgraph based on the boolean value of the tag never_reuse
  - PR#4039 - fixes issue#3745 : Made the GPU OPs use _props_dict() during all possible GPU optimization. This helps to detect future error if we add props to the CPU but forget to update the GPU optimizations and op.
  - PR#3983- fixes issue#3900 : Added required __future__ module to all the necessary places across the codebase.

Project Info:

- **Proposal title : Theano: Better handling of large graph and faster optimization during compilation**
- **Proposal Abstract :**

  This project aims at improvising graph traversal and serialization of objects, optimizations on GPU, optimization of fast_run, inclusion of *optimizer_excluding* flag, fixing of slow optimizing phase during compilation and faster cyclic detection.
- **Proposal Detailed Description :**

  After going through the issues in detail, analyzing how CGT(Computational Graph Toolkit) overcomes these issue, discussing with my mentor Frédéric, I propose to work on the following :

Zeroth Phase : **Profiling and Analyzing**

The tasks proposed in this phase will be performed before the coding phase begins. Ideally, i am planning to start right after the proposal submission period. I ought to have done some of these profiling before, but wasn't able to as I was in the middle of my mid-term examinations.

1. Various profiling will be done and the slow optimizations will be tagged as slow. This is done in-order to have an intermediate between fast_compile and fast_run.
2. Big graphs will be profiled and optimization phases that are time consuming will be collected. The profiled big graphs will be a collection of the real use cases, collected from theano-user group.
3. Profiling will be done using CGTOptimizer and the change in the optimizations manner and order of optimizations will be recorded in-order to modify the current optimizations to work in that fashion.
4. A hacky version of faster cycle detection:
   4.1. *"io_toposort"* will be called on fgraph and the check for cyclic fgraph will be made from inside a loop to speed up the algorithm to O(N)
   4.2. The profiling for the altered algorithm will be done on scan and convolution. (on GPU)

First phase : Fix recursion and Improvise serialization of objects

1. Because of the use of recursive algorithms, theano makes python reach the maximum stack limit and thus could not handle large and deep graphs. There are two recursions that has to be broken, while handling graph traversal and from the output-input and the input-output.

2. For breaking of recursion while graph traversal, an approach similar to CGT will be followed. *stack_search* method will be used for a stack-based Depth First Search of the graph, transforming

the existing recursive algorithm to a stack-based algorithm. This will be applied across "rebuild_collect_shared" and other similar methods that uses recursion.

3. For the methods *"pre_greedy_local_optimizer"*, "*recursive_merge*", a maximum fixed recursion limit of depth 50 would be applied.

4. The recursion between output-input and input-output will be broken down by reordering deepcopy function of FunctionGraph and deleting the client fields of Variable, rebuilding the graph respectively.

5. An alternative approach would be to change the way client is being represented. A dictionary(clients_dict) will be made in FunctionGraph class that has key as an ApplyNode and value is a client list. To not break the interface, the Variable.clients will me made as a property that checks and returns the clients_dict dictionary of the FunctionGraph.

6. I previously did some groundwork, testing out various possible alternative libraries(cloudpickle, dll), to pickle and tested with the same pickle on python3 by compiling a large graph. An example that failed to pickle is attached in the reference section. The new change that i propose to do is,
    1.1. The __*getstate*__ and __*setstate*__ methods in the function graph will be overridden. The client fields in *variable* set will be removed using the __*getstate*__ method and will be recreated from the __*setstate*__ method.

2. Estimated time : 2 weeks

Second Phase : Global optimization that moves computation to the GPU

1. Currently, for optimization on GPU, the user builds a graph on the CPU and the existing optimizations move that to GPU. A global optimizer will be created that would use *io_toposort* to find the order of operation and do the move to GPU in one pass. The global optimizer will make use of the current optimizer to move the required optimizations to the GPU. A backward pass will also be performed to ensure whether the op in the middle of the graph is implemented on the GPU.

2. Estimated time : 1 week

Third Phase : Adding flag Optimizer_excluding=slow

1. A new flag will be introduced for slow optimizations. The groundwork for tagging the slow optimizations would have been completed before the coding period began. And all the optimizations collected before, that happen to take a longer time for compilation will be applied when execution is performed with this flag. The number of slow optimizations to be moved into slow tag is tentative and will be discussed as the project proceeds.

2. Estimated time : 1 week

Fourth Phase : Optimization on *fast_run* mode.

1. The goal of this phase is to make EquilibriumOptimizer loop less.
2. Based on the observations noted from the profiling done with CGTOptimizer before the coding period, the EquilibriumOptimizer will be replaced with CGTOptimizer's algorithm and will be applied to canonicalize, stabilize and specialize phases (in the given order).

3. **Estimated time : 2 weeks**

Fifth Phase : Fixing slow optimization during compiling

1. The optimization phases that has longer compilation time (which was already collected) will be removed or improvised.
2. The optimization phases to be speeded up is tentative. The profiling will be done again and will discuss with mentors to finalize upon which optimizations are to be speeded up.
3. **Estimated time : 2 weeks**

Sixth Phase : Faster Cycle detection during optimization

1. Currently, before inplace optimizations are performed, a Feature is introduced, that performs validation to not allow cycles in the graph. The goal of this phase is to speed up this validation.
2. Initial speeding up would be done to O(N) during the zeroth phase, before the coding period. This O(N) will be improvized to O(1) during this phase.
3. For speeding up, the graph would not be iterated every time, whenever there has been a change in the graph.
4. This will be done by doing the validation in *on_import()* and *on_input_change()* method, where, the errors in the Feature are recorded.
5. Then in the method *validate*, an InconsistencyError will be raised. This allow to only verify the changed part of the graph. So the verification will be constant time in the size of the graph.
6. Generalize the method implemented to sequence inplace/view that has one client.
7. For further boost in the speed up, the OrderedDict used in *orderings()* will be replaced with DefaultDict. Since we only query the dict, it doesn't need to be ordered.
8. If time permits, I will implement the full cycle detection in cython with ctypes.
9. **Estimated time : 3 weeks**

Project timeline :

Pre-GSoC: Till 22nd May	Zeroth Phase : Profiling and analyzing
23rd May - 6th June	First phase : Fix recursion and Improvise serialization of objects
7th June - 13thJune	Second Phase : Global optimization that moves computation to the GPU
14th June - 20th June	Third Phase : Adding flag Optimizer_excluding=slow

21st June	Cleaning up and submitting the code for mid-term evaluation
22nd June - 5th July	Fourth Phase : Optimization for *fast_run* mode.
6thJuly - 19th July	Fifth Phase : Fixing slow optimization during compiling
20th July - 8thAugust	Sixth Phase : Faster Cycle detection during optimization
9th August - 16th August	Reserve week. If any work is left undone or half done or postponed, it would be covered in this week.

Other Commitments:

I would be applying for Deep Learning Summer school in the first week of August. In case I happen to attend the summer school(which is a week long program), I promise to complete that week's work before this program ends.

Apart from this I have absolutely no other commitments during the GSoC period and will dedicate my full concentration on this project.

I would take every Sunday off but will maintain the minimum of 40hrs per week of work.

If required by mentors, I will make the required changes to my proposal post submission.

Also, I have not applied for any other organizations.

Extra information:

- Link to resume : Resume

- University Name: Birla Institute of Technology and Science, Goa, India
- Current Year of Study : 3rd year
- Expected Graduation Year : 2017
- Degree: B.E(Hons)
- Instant Messaging information
    - Skype: vxrram95
    - Gtalk: vxrram95@gmail.com
    - Yahoo Messenger: vxr95
- Twitter - https://twitter.com/ramana_95

References:

[1] https://gist.github.com/Sentient07/61937a7cbd65e594e10c - A large graph that fails on theano, successfully complies on CGT

[2] https://gist.github.com/Sentient07/1d89790afd6e67c68bb3 - Failing to pickle large objects by the different modules initially proposed.

[3] http://stackoverflow.com/questions/19629682/ordereddict-vs-defaultdict-vs-dict - Speed of OrderedDict vs DefaultDict

[4] https://github.com/joschu/cgt

Issues addressed :

[5] #1681 , #4233, #2470, #4275, #2494