GPU processing with Theano

Razvan Pascanu (Google DeepMind)

Theano

Google DeepMind

Personal Info

- **Masters:**  JACOBS UNIVERSITY in Bremen, Germany
- **Advisor:** Herbert Jaeger 

  Source: `http://minds.jacobs-university.de/herbert.html`
- **Topic:** Memory-like behaviour in recurrent neural networks

Theano

Google DeepMind

Personal Info

- **PhD:** Université de Montréal, Canada
- **Advisor:** Yoshua Bengio 

  Source: `http://www.iro.umontreal.ca/~bengioy/yoshua_en/index.html`
- **Topics:**
  - Recurrent Neural Networks (e.g. learning dynamics/memory/etc)
  - Optimization for neural networks (e.g. natural gradient/error landscape)
  - Theory for neural networks (e.g. deep vs shallow)
  - Theano (e.g. looping/conditional computation)

Theano

Google DeepMind

Personal Info

- **Job:** Researcher at 

, London, UK
- **Topics:**
  - Optimization
  - Reinforcement Learning and Deep Learning
  - Deep Learning
  - Generative Models
  - etc.
- **Publications:** http://deepmind.com/publications.html

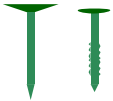
Now a bit about yourself

- How many of you played with **MLPs / convolutional nets**?
- How many of you played with **RNNs** ?
- How many used **Torch, Caffe**, or, Theano indirectly (**pylearn2, blocks, lasagne**)?
- How many already know **Theano** ?
- How many know about **Reinforcement Learning (Q-learning)** ?

Theano

Google DeepMind

Software infrastructure: Why?

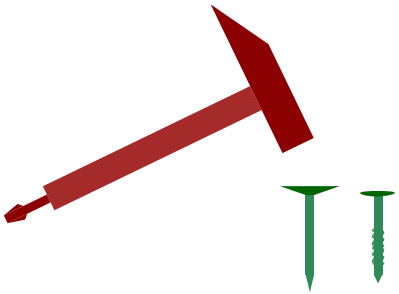


- Neural networks become interesting in the *large data regime*
- Learning can be a *slow* process
- Neural networks are a *flexible* paradigm

Theano

Google DeepMind

Software infrastructure: Why?



To be competitive (or even keep up) you need **a good hammer**.

- Solid software infrastructure
- Efficiency (on new hardware) with minimal effort
- Flexibility
- Simple interface
- Lots of coffee

Theano

Google DeepMind

And **Theano** does all of this (except the coffee ..).

Theano overview

pros	cons
▶ extremely flexible ▶ symbolic differentiation ▶ efficient ▶ transparent GPU/CPU (stabilization tricks) ▶ intuitive syntax ▶ python interface compatible with numpy/scipy	▶ steep learning curve ▶ different phisolophy of coding ▶ "unpredictable" slowness ▶ requires compilation at runtime (g++, nvcc)

Theano

Google DeepMind

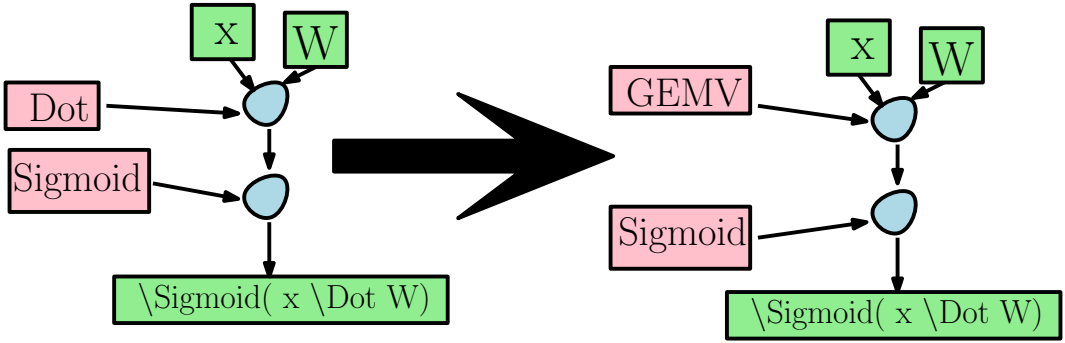
Theano and extensions

- **PyLearn2** (`https://github.com/lisa-lab/pylearn2`)
- **Blocks** (`https://github.com/mila-udem/blocks`)
- **Lasagne** (`https://github.com/Lasagne/Lasagne`)

Theano

Google DeepMind

Basics of Theano:Philosophy



Theano

Google DeepMind

Basics of Theano:Simple Example

```
import numpy
import theano
import theano.tensor as TT

x = TT.vector("input")
W = TT.shared(numpy.random.uniform(
        low=.01,
        high=.01,
        size=(500,784)))
out = TT.tanh(TT.dot(W, x))

fn = theano.function(x, out)
```

Theano

Google DeepMind

Basics of Theano:Variables

```
x = TT.fvector("input")
W = TT.shared(numpy.random.uniform(
        low=.01,
        high=.01,
        size=(500,784)).astype('float32'))
```

Theano

Google DeepMind

Basics of Theano: Variables

theano.compile.sharedvalue.**shared**(*value*, *name=None*, *strict=False*, *allow_downcast=None*, ***kwargs*)

Return a SharedVariable Variable, initialized with a copy or reference of `value`.

This function iterates over *constructor functions* to find a suitable SharedVariable subclass. The suitable one is the first constructor that accept the given value.

This function is meant as a convenient default. If you want to use a specific shared variable constructor, consider calling it directly.

`theano.shared` is a shortcut to this function.

Note By passing kwargs, you effectively limit the set of potential constructors to those that can accept those kwargs.
:

Note Some shared variable have `borrow` as extra kwargs. See for detail.
:

Note Some shared variable have `broadcastable` as extra kwargs. As shared variable shapes can change, all dimensions default to not being broadcastable, even if `value` has a shape of 1 along some
: dimension. This parameter allows you to create for example a *row* or *column* 2d tensor.

Source: Theano Library documentation

Basics of Theano: Operations

```
out = TT.tanh(TT.dot(W, x))
```

Theano

Google DeepMind

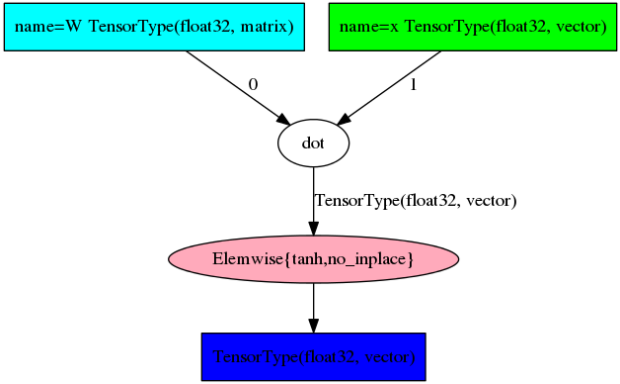
Basics of Theano: Operations

- TT.Dot
- TT.tanh
- TT.nnet.sigmoid
- TT.nnet.relu
- TT.nnet.conv2d
- * + / -
- ...

Theano

Google DeepMind

Basics of Theano: Operations



Theano

Google DeepMind

Basics of Theano: Function

```
fn = theano.function(x, out)
```

Theano

Google DeepMind

Basics of Theano: Function

`function.`**`function`**(*inputs*, *outputs*, *mode=None*, *updates=None*, *givens=None*, *no_default_updates=False*, *accept_inplace=False*, *name=None*, *rebuild_strict=True*, *allow_input_downcast=None*, *profile=None*, *on_unused_input='raise'*)

Return a callable object that will calculate *outputs* from *inputs*.

Parameters:
- **params** (*list of either Variable or Param instances, but not shared variables.*) – the returned `Function` instance will have parameters for these variables.
- **outputs** (*list of Variables or Out instances*) – expressions to compute.
- **mode** (None, string or `Mode` instance.) – compilation mode
- **updates** (*iterable over pairs (shared_variable, new_expression). List, tuple or dict.*) – expressions for new `SharedVariable` values
- **givens** (*iterable over pairs (Var1, Var2) of Variables. List, tuple or dict. The Var1 and Var2 in each pair must have the same Type.*) – specific substitutions to make in the computation graph (Var2 replaces Var1).
- **no_default_updates** (*either bool or list of Variables*) – if True, do not perform any automatic update on Variables. If False (default), perform them all. Else, perform automatic updates on all Variables that are neither in `updates` nor in `no_default_updates`.
- **name** – an optional name for this function. The profile mode will print the time spent in this function.
- **rebuild_strict** – True (Default) is the safer and better tested setting, in which case *givens* must substitute new variables with the same Type as the variables they replace. False is a you-better-know-what-you-are-doing setting, that permits *givens* to replace variables with new variables of any Type. The consequence of changing a Type is that all results depending on that variable may have a different Type too (the graph is rebuilt from inputs to outputs). If one of the new types does not make sense for one of the Ops in the graph, an Exception will be raised.
- **allow_input_downcast** (*Boolean or None*) – True means that the values passed as inputs when calling the function can be silently downcasted to fit the dtype of the corresponding Variable, which may lose precision. False means that it will only be cast to a more general, or precise, type. None (default) is almost like False, but allows downcasting of Python float scalars to floatX.
- **profile** (*None, True, or ProfileStats instance*) – accumulate profiling information into a given ProfileStats instance. If argument is *True* then a new ProfileStats instance will be used. This profiling object will be available via self.profile.
- **on_unused_input** – What to do if a variable in the 'inputs' list is not used in the graph. Possible values are 'raise', 'warn', and 'ignore'.

Return type: Function instance

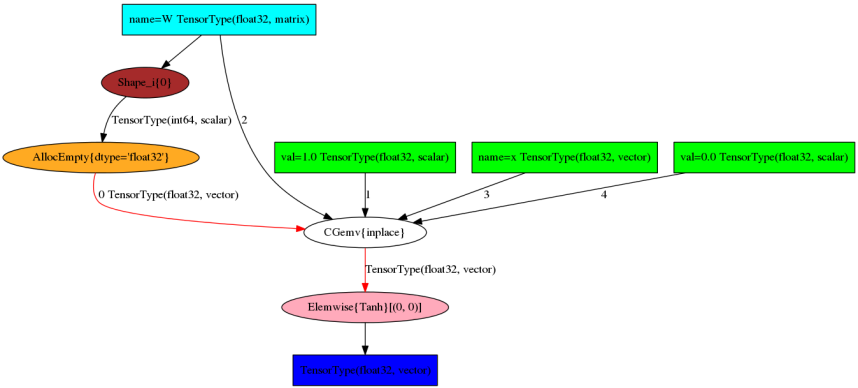
Returns: a callable object that will compute the outputs (given the inputs) and update the implicit function arguments according to the *updates*.

Source: Theano Library documentation

Theano

Google DeepMind

Basics of Theano: Using a function



Theano

Google DeepMind

Basics of Theano: Using a function

```
x = numpy.random.uniform(size=(784,)).astype('float32')
print (fn(x))
```

Theano

Google DeepMind

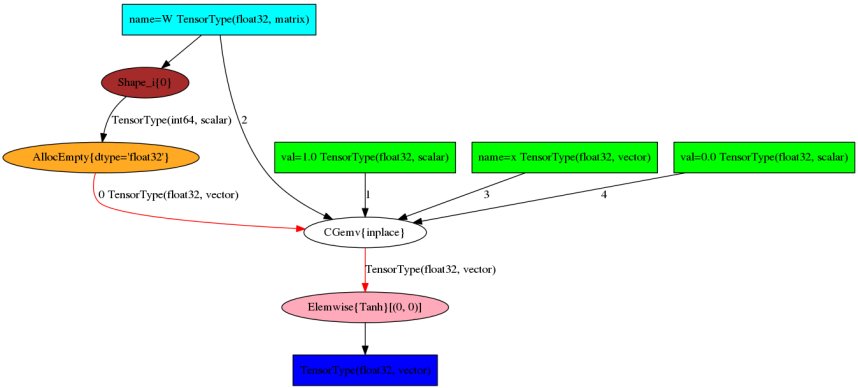
Basics of Theano: Grad

```
gW = TT.grad(out.sum(), W)
```

Theano

Google DeepMind

Basics of Theano: Grad



Theano

Google DeepMind

Basics of Theano: Grad

theano.gradient.**grad**(*cost*, *wrt*, *consider_constant=None*, *disconnected_inputs='raise'*, *add_names=True*, *known_grads=None*, *return_disconnected='zero'*)

Return symbolic gradients for one or more variables with respect to some cost.

For more information about how automatic differentiation works in Theano, see `gradient`. For information on how to implement the gradient of a certain Op, see `grad()`.

Parameters:
- **cost** (*Scalar (0-dimensional) tensor variable. May optionally be None if known_grads is provided.*) – a scalar with respect to which we are differentiating
- **wrt** (*Tensor variable or list of variables.*) – term[s] for which we want gradients
- **consider_constant** (*list of variables*) – a list of expressions not to backpropagate through
- **disconnected_inputs** (*string*) – Defines the behaviour if some of the variables in `wrt` are not part of the computational graph computing `cost` (or if all links are non-differentiable). The possible values are: - 'ignore': considers that the gradient on these parameters is zero. - 'warn': consider the gradient zero, and print a warning. - 'raise': raise DisconnectedInputError.
- **add_names** (*bool*) – If True, variables generated by grad will be named (d<cost.name>/d<wrt.name>) provided that both cost and wrt have names
- **known_grads** (*dict*) – If not None, a dictionary mapping variables to their gradients. This is useful in the case where you know the gradient on some variables but do not know the original cost.
- **return_disconnected** (*string*) –
  - 'zero' : *If wrt[i] is disconnected, return value i will be*
    wrt[i].zeros_like()
  - 'None' : *If wrt[i] is disconnected, return value i will be*
    None
  - 'Disconnected' : returns variables of type DisconnectedType

Return type: variable or list/tuple of Variables (matching *wrt*)

Returns: symbolic expression of gradient of *cost* with respect to each of the *wrt* terms. If an element of *wrt* is not differentiable with respect to the output, then a zero variable is returned. It returns an object of same type as *wrt*: a list/tuple or Variable in all cases.

Source: Theano Library documentation

Basics of Theano: Scan I

$$x(n) = \tanh\left(Wx(n-1) + W_1^{in}u(n) + W_2^{in}u(n-4) + W^{feedback}y(n-1)\right)$$

$$y(n) = W^{out}x(n-3)$$

Theano

Google DeepMind

Basics of Theano: Scan II

```
def oneStep(u_tm4, u_t, x_tm3, x_tm1, y_tm1, W, W_in_1,
            W_in_2,  W_feedback, W_out):

  x_t = T.tanh(theano.dot(x_tm1, W) +
               theano.dot(u_t,   W_in_1) +
               theano.dot(u_tm4, W_in_2) +
               theano.dot(y_tm1, W_feedback))
  y_t = theano.dot(x_tm3, W_out)

  return [x_t, y_t]
```

Theano

Google DeepMind

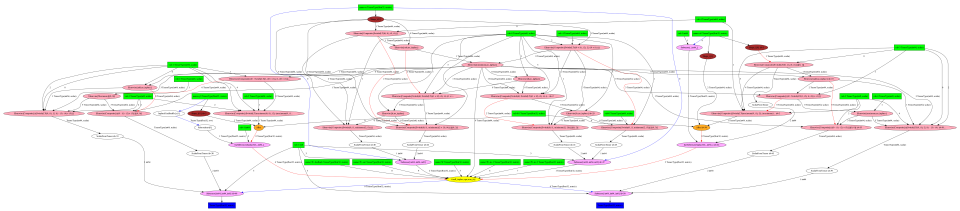
Basics of Theano: Scan III

```
([x_vals, y_vals], updates) = theano.scan(fn=oneStep,
      sequences=dict(input=u, taps=[-4,-0]),
      outputs_info=[dict(initial=x0, taps=[-3,-1]), y0],
      non_sequences=[W, W_in_1, W_in_2, W_feedback, W_out],
      strict=True)
    # for second input y, scan adds -1 in
    # output_taps by default
```

Theano

Google DeepMind

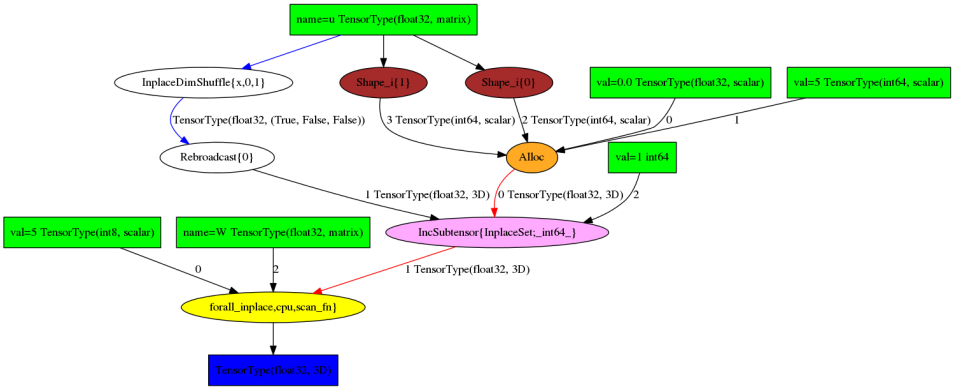
Basics of Theano: Scan IV



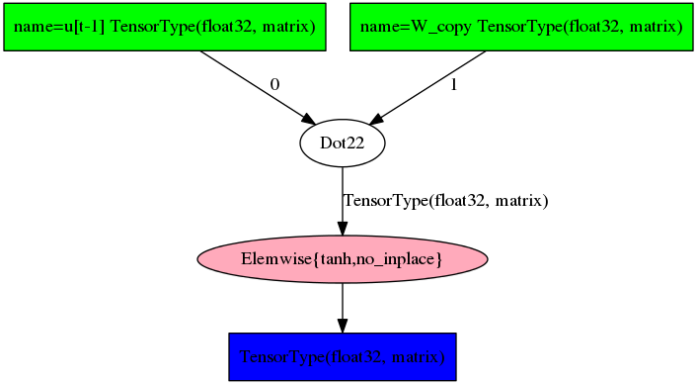
Theano

Google DeepMind

Basics of Theano: Scan V



Basics of Theano: Scan VI



Theano

Google DeepMind

Basics of Theano: Scan VII

```
theano.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)
```

Theano

Google DeepMind

To be continued ...

The magic of GPUs

Source: `http://docs.nvidia.com/cuda/cuda-c-programming-guide`

ALU = Arithmetic Logic Units (workhorse for arithmetic and bitwise logical operations)

Theano

Google DeepMind

- The power is in having **more** cores rather than in faster cores.

Comparing GPUs

SELECT THE RIGHT TESLA GPU

Features	Tesla K80[1]	Tesla K40
GPU	2x Kepler GK210	1 Kepler GK110B
Peak double precision floating point performance	2.91 Tflops (GPU Boost Clocks) 1.87 Tflops (Base Clocks)	1.66 Tflops (GPU Boost Clocks) 1.43 Tflops (Base Clocks)
Peak single precision floating point performance	8.74 Tflops (GPU Boost Clocks) 5.6 Tflops (Base Clocks)	5 Tflops (GPU Boost Clocks) 4.29 Tflops (Base Clocks)
Memory bandwidth (ECC off)[2]	480 GB/sec (240 GB/sec per GPU)	288 GB/sec
Memory size (GDDR5)	24 GB (12GB per GPU)	12 GB
CUDA cores	4992 (2496 per GPU)	2880

Source: `www.nvidia.com/object/tesla-servers.html`

With (k80) a max frequency per core of just 875 MHz

Theano

Google DeepMind

Shift in paradigm

Around 1999/2000:

- **Fallout**
- **Tomb Raider II**
- **Quake II**
- **...**

What I was doing:

Theano

Google DeepMind

Shift in paradigm

Around 1999/2000:

- **Fallout**
- **Tomb Raider II**
- **Quake II**
- **...**

What I was doing:

Theano

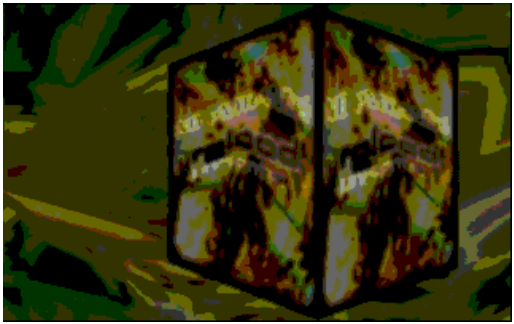
Google DeepMind

Shift in paradigm

Around 1999/2000:

- **Fallout**
- **Tomb Raider II**
- **Quake II**
- **...**

What I was doing:



Theano

Google DeepMind

Shift in paradigm

Around 1999/2000:

- **Fallout**
- **Tomb Raider II**
- **Quake II**
- **...**

What I was doing:

Theano

Google DeepMind

Shift in paradigm

Around 1999/2000:

- **Fallout**
- **Tomb Raider II**
- **Quake II**
- **...**

What I was doing:

Theano

Google DeepMind

Shift in paradigm

Around 1999/2000:

- **Fallout**
- **Tomb Raider II**
- **Quake II**
- **...**

What I was doing:

Theano

Google DeepMind

Shift in paradigm

Now also heavily used for machine learning (deep learning)

DIGITS

Interactive deep learning GPU training system for image classification

cuDNN

GPU-accelerated deep neural network library accelerates widely-used frameworks

CUDA Toolkit

Complete C/C++ SDK with libraries for deep learning researchers and developers

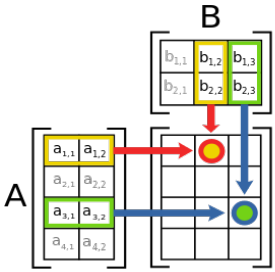
Source: `https://developer.nvidia.com/deep-learning`

Allmost any deep learning success story hides behind hundreds of GPUs.

Theano

Google DeepMind

Efficiency of parallelization (e.g. matrix multiplication)



Source: https://en.wikipedia.org/wiki/Matrix_multiplication

Virtually, GPUs can make an $O(nm)$ into $O(m)$ algorithm.

Theano

Google DeepMind

Theano and GPUs

Go to http://deeplearning.net/software/theano/install.html

- Define a $CUDA_ROOT environment variable to equal the cuda root directory, as in CUDA_ROOT=/path/to/cuda/root, or
- add a cuda.root flag to THEANO_FLAGS, as in THEANO_FLAGS='cuda.root=/path/to/cuda/root', or
- add a [cuda] section to your .theanorc file containing the option root = /path/to/cuda/root.

Theano

Google DeepMind

Basics of Theano on GPU: Flags

```
THEANO_FLAGS=floatX=float32,device=gpu0 python \
    my_script.py
```

And it should work?

Theoretically yes ... practically not quite.

How does GPU code work?

For most Ops there exists a GPU variant:

- Gemm $\Leftrightarrow$ GpuGemm
- Softmax $\Leftrightarrow$ GpuSoftmax
- ...

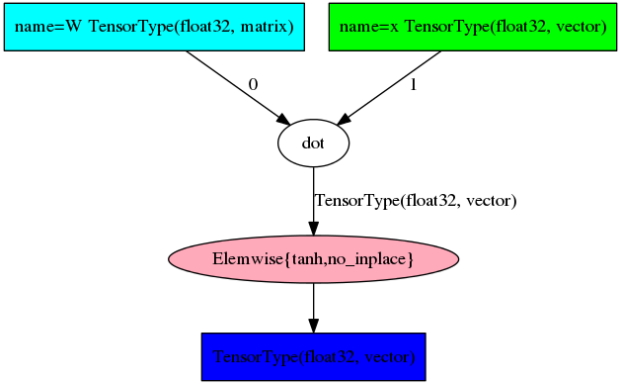
You have GpuFromHost and HostFromGpu to move data around.

- **Optimizations** will try to move ops from CPU to GPU (patching with GpuFromHost, HostFromGpu where needed)

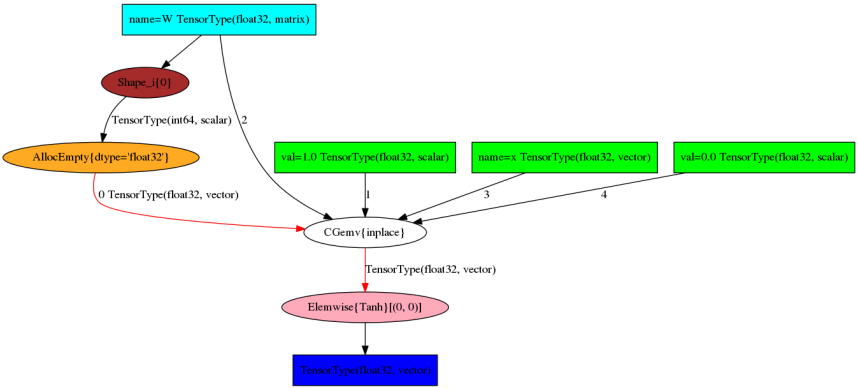
Theano

Google DeepMind

How does GPU code work?



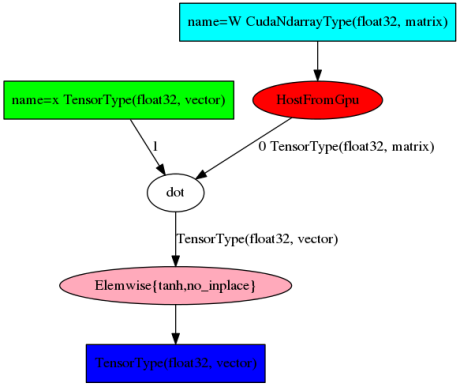
How does GPU code work?



Theano

Google DeepMind

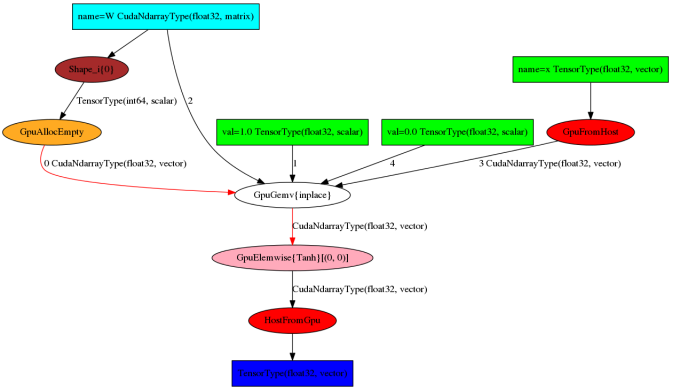
How does GPU code work?



Theano

Google DeepMind

How does GPU code work?



How does a GPU Op work?

```
def c_code(self, node, nodename, inputs, outputs, sub):
    return """
    if (%(x)s->nd != 2)
    ...
     if (CudaNdarray_gemm(1.0f, %(x)s, %(y)s, 0.0f, %(z)s))
        {
         if (%(z)s)
         {
          Py_DECREF(%(z)s);
    ...
    """ % locals();
```

Theano

Google DeepMind

Debugging tools: Print Op

```
h = theano.printing.Print("value x",
                          attrs=("mean", "max"))(h)
```

Debugging tools: debugprint

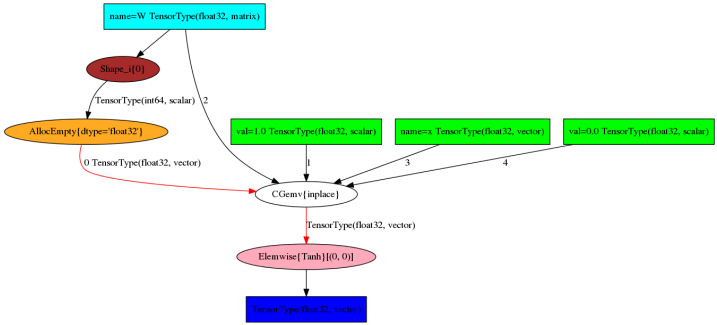
```
> theano.printing.debuprint(prediction)
 Elemwise{gt,no_inplace} [@A] ''
   |Elemwise{true_div,no_inplace} [@B] ''
   | |DimShuffle{x} [@C] ''
   | | |TensorConstant{1} [@D]
   | |Elemwise{add,no_inplace} [@E] ''
   |   |DimShuffle{x} [@F] ''
   |   | |TensorConstant{1} [@D]
   |   |Elemwise{exp,no_inplace} [@G] ''
   |     |Elemwise{sub,no_inplace} [@H] ''
   |       |Elemwise{neg,no_inplace} [@I] ''
   |       | |dot [@J] ''
```

Theano

Google DeepMind

Debugging tools: pydotprint

```
> theano.printing.pydotprint(func)
```



Debugging tools: Test values

```
# enable on-the-fly graph computations
theano.config.compute_test_value = 'warn'

...

# input which will be of shape (5, 10)
x  = T.matrix('x')
# provide Theano with a default test-value
x.tag.test_value = numpy.random.rand(5, 10)
```

Theano

Google DeepMind

Debugging tools: ipdb

```
import ipdb

...
ipdb.set_trace()
```

Theano

Google DeepMind

Configuring Theano: MODE

```
fun = theano.function([x], out, mode='DebugMode')
```

short name	Full constructor	What does it do?
FAST_COMPILE	compile.mode.Mode(linker='py', optimizer='fast_compile')	Python implementations only, quick and cheap graph transformations
FAST_RUN	compile.mode.Mode(linker='cvm', optimizer='fast_run')	C implementations where available, all available graph transformations.
DebugMode	compile.debugmode.DebugMode()	Both implementations where available, all available graph transformations.
ProfileMode	compile.profilemode.ProfileMode()	Deprecated. C implementations where available, all available graph transformations, print profile information.

Theano

Google DeepMind

Configuring Theano: Linker

```
fun = theano.function([x], out,
    mode=theano.mode.Mode(linker='cvm',
    optimizer='fast_run')
```

linker	gc [1]	Raise error by op	Overhead	Definition
cvm	yes	yes	"++"	As c|py, but the runtime algo to execute the code is in c
cvm_nogc	no	yes	"+"	As cvm, but without gc
c|py [2]	yes	yes	"+++"	Try C code. If none exists for an op, use Python
c|py_nogc	no	yes	"++"	As c|py, but without gc
c	no	yes	"+"	Use only C code (if none available for an op, raise an error)
py	yes	yes	"+++"	Use only Python code
ProfileMode	no	no	"++++"	(Deprecated) Compute some extra profiling info
DebugMode	no	yes	VERY HIGH	Make many checks on what Theano computes

Theano

Google DeepMind

Configuring Theano: Profiling

```
fun = theano.function([x], out, profile=True)
```

```
Function profiling
==================
  Message: None
  Time in 1 calls to Function.__call__: 5.698204e-05s
  Time in Function.fn.__call__: 1.192093e-05s (20.921%)
  Time in thunks: 6.198883e-06s (10.879%)
  Total compile time: 3.642474e+00s
    Theano Optimizer time: 7.326508e-02s
       Theano validate time: 3.712177e-04s
    Theano Linker time (includes C, CUDA code generation/compiling): 9.584920e-01s

Class
---
<% time> <sum %> <apply time> <time per call> <type> <#call> <#apply> <Class name>
  100.0%   100.0%       0.000s       2.07e-06s     C       3       3   <class 'theano.tensor.elemwise.Elemwise'>
   ... (remaining 0 Classes account for   0.00%(0.00s) of the runtime)

Ops
---
<% time> <sum %> <apply time> <time per call> <type> <#call> <#apply> <Op name>
  65.4%    65.4%       0.000s       2.03e-06s     C       2       2   Elemwise{add,no_inplace}
  34.6%   100.0%       0.000s       2.15e-06s     C       1       1   Elemwise{mul,no_inplace}
   ... (remaining 0 Ops account for   0.00%(0.00s) of the runtime)

Apply
------
<% time> <sum %> <apply time> <time per call> <#call> <id> <Apply name>
  50.0%    50.0%       0.000s       3.10e-06s      1     0   Elemwise{add,no_inplace}(x, y)
  34.6%    84.6%       0.000s       2.15e-06s      1     2   Elemwise{mul,no_inplace}(TensorConstant{(1,) of 2.0}, Elemwise{add,no_inplace}.0)
  15.4%   100.0%       0.000s       9.54e-07s      1     1   Elemwise{add,no_inplace}(Elemwise{add,no_inplace}.0, z)
   ... (remaining 0 Apply instances account for 0.00%(0.00s) of the runtime)
```

Theano

Google DeepMind

Carefully writing your graph !?

Theano is helpful but it is no magic

- If code is slow, **profile** !
- Ensure you do not have many HostFromGpu, GpuFromHost nodes
- Ask for help and be patient
- Optimization is NP hard so give it some slack
- Rely on the many existing examples ...

Theano

Google DeepMind

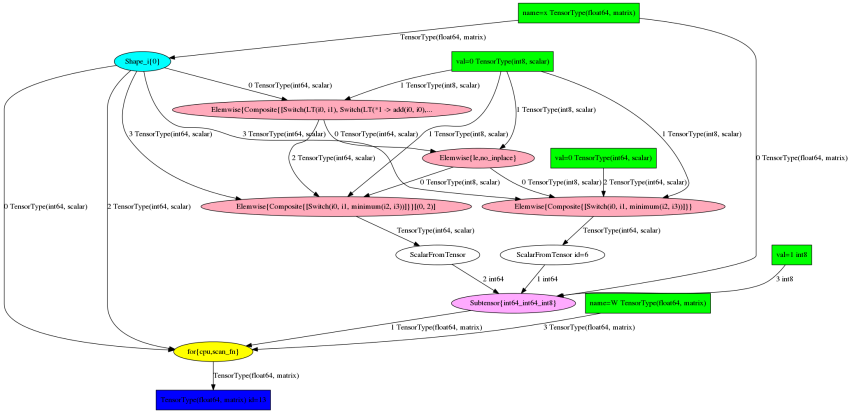
Carefully writing your graph – example

```
x = TT.matrix('x')
W = TT.matrix('W')
out, _ = theano.scan(lambda x,W:TT.dot(W,x),
                     sequences=x,
                     non_sequences=W)
```

Theano

Google DeepMind

This should be just a GEMM call ...



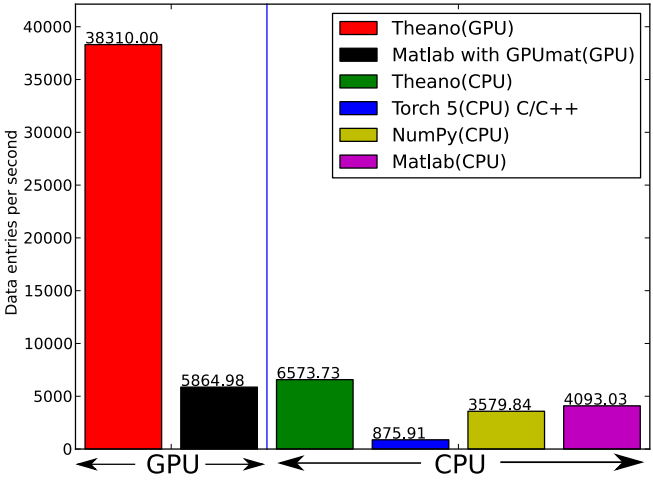
GPU: words of wisdom

- only *float32*
- only matrix multiplications, convolutions, large element-wise operations will be accelerated
- indexing, dimension-shuffling as fast on CPU as GPU
- summation over rows/cols slightly slower
- copying large amounts of data is slow
- set *allow_gc=False*

Theano

Google DeepMind

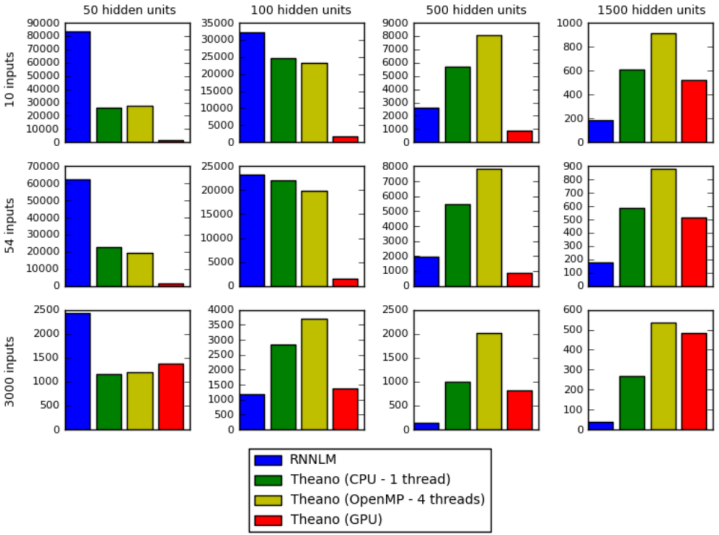
Benchmarks



Theano

Google DeepMind

Benchmarks



Theano

Google DeepMind

Citing Theano

- F. Bastien, P. Lamblin, R. Pascanu, J. Bergstra, I. Goodfellow, A. Bergeron, N. Bouchard, D. Warde-Farley and Y. Bengio. *Theano: new features and speed improvements*. NIPS 2012 deep learning workshop.
- J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley and Y. Bengio. *Theano: A CPU and GPU Math Expression Compiler*. Proceedings of the Python for Scientific Computing Conference (SciPy) 2010. June 30 - July 3, Austin, TX

Theano

Google DeepMind

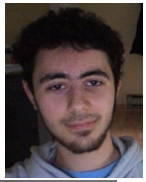
How to get help

- Register to theano-announce for important changes
- Register to theano-users for help
- Register to theano-dev for devs.
- Ask/view questions/answers at StackOverflow
- Look at Github tickets

Theano

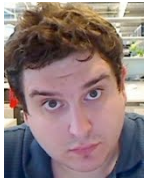
Google DeepMind

Developers (early days)





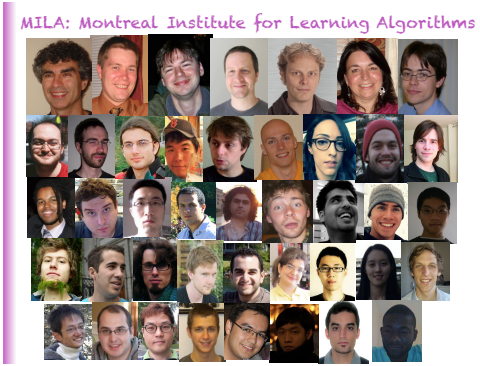



Theano

Google DeepMind

Developers

MILA students, from where many of the new Theano dev team come from:



i

Source: http://www.iro.umontreal.ca/~bengioy/talks/NIPS-OPT-workshop-12dec2014.pdf

Theano

But also many many more ...

Google DeepMind

Thank you

Questions ?

Theano

Google DeepMind

Possible exercise for the afternoon sessions I

Pick one or several tasks from the Deep Learning Tutorials:

- Logistic Regression
- MLP
- AutoEncoders / Denoising AutoEncoders
- Stacked Denoising AutoEncoders

Theano

Google DeepMind

Possible exercise for the afternoon sessions II

Compare different initialization for neural networks (MLPs and ConvNets) with rectifieres or tanh. In particular compare:

- The initialization proposed by Glorot et al.
- Sampling uniformally from $[-\frac{1}{fan_{in}}, \frac{1}{fan_{in}}]$
- Setting all singular values to 1 (and biases to 0)

How do different optimization algorithms help with this initializations? Extra kudos for interesting plots or analysis. Please make use of the Deep Learning Tutorials.

Theano

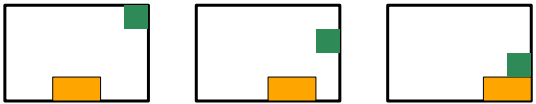
Google DeepMind

Possible exercise for the afternoon sessions III
Requires convolutions

Re-implement the AutoEncoder tutorial using convolutions both in the encoder and the decoder. Extra kudos for allowing pooling (or strides) in the encoder.

Possible exercise for the afternoon sessions IV
Requires Reinforcement Learning

Attempt to solve the Catch game.



Not actual screenshots of the game

Theano

Google DeepMind