# Theano目录

1、[Theano1.1-安装](http://www.cnblogs.com/shouhuxianjian/p/4564630.html)

2、[Theano2.1.1-基础知识之准备工作](http://www.cnblogs.com/shouhuxianjian/p/4564622.html)

3、[Theano2.1.2-基础知识之第一步：代数](http://www.cnblogs.com/shouhuxianjian/p/4590234.html)

4、[Theano2.1.3-基础知识之更多的例子](http://www.cnblogs.com/shouhuxianjian/p/4594517.html)

5、[Theano2.1.4-基础知识之图结构](http://www.cnblogs.com/shouhuxianjian/p/4590233.html)

6、[Theano2.1.5-基础知识之打印出theano的图](http://www.cnblogs.com/shouhuxianjian/p/4590231.html)

7、[Theano2.1.6-基础知识之在thenao中的求导](http://www.cnblogs.com/shouhuxianjian/p/4590230.html)

8、[Theano2.1.7-基础知识之设置的配置和编译模式](http://www.cnblogs.com/shouhuxianjian/p/4590229.html)

9、[Theano2.1.8-基础知识之装载和保存](http://www.cnblogs.com/shouhuxianjian/p/4590228.html)

10、[Theano2.1.9-基础知识之条件](http://www.cnblogs.com/shouhuxianjian/p/4590227.html)

11、[Theano2.1.10-基础知识之循环](http://www.cnblogs.com/shouhuxianjian/p/4590226.html)

12、[Theano2.1.11-基础知识之稀疏](http://www.cnblogs.com/shouhuxianjian/p/4590225.html)

13、[Theano2.1.12-基础知识之使用GPU](http://www.cnblogs.com/shouhuxianjian/p/4590224.html)

14、[Theano2.1.13-基础知识之PyCUDA、CUDAMat、Gnumpy的兼容](http://www.cnblogs.com/shouhuxianjian/p/4590223.html)

15、[Theano2.1.14-基础知识之理解为了速度和正确性的内存别名](http://www.cnblogs.com/shouhuxianjian/p/4590222.html)

16、[Theano2.1.15-基础知识之theano如何处理shapre信息](http://www.cnblogs.com/shouhuxianjian/p/4590221.html)

17、[Theano2.1.16-基础知识之调试：常见的问题解答](http://www.cnblogs.com/shouhuxianjian/p/4590220.html)

18、[Theano2.1.17-基础知识之剖析theano的函数](http://www.cnblogs.com/shouhuxianjian/p/4590219.html)

19、[Theano2.1.18-基础知识之theano的扩展](http://www.cnblogs.com/shouhuxianjian/p/4594532.html)

20、

21、

22、[Theano2.1.21-基础知识之theano中多核的支持](http://www.cnblogs.com/shouhuxianjian/p/4594524.html)

23、[Theano3.1-练习之初步介绍](http://www.cnblogs.com/shouhuxianjian/p/4564623.html)

24、[Theano3.2-练习之数据集及目标函数介绍](http://www.cnblogs.com/shouhuxianjian/p/4564627.html)

25、[Theano3.3-练习之逻辑回归](http://www.cnblogs.com/shouhuxianjian/p/4564626.html)

26、[Theano3.4-练习之多层感知机](http://www.cnblogs.com/shouhuxianjian/p/4564625.html)

27、[Theano3.5-练习之深度卷积网络](http://www.cnblogs.com/shouhuxianjian/p/4564624.html)

28、

# [Theano1.1-安装](http://www.cnblogs.com/shouhuxianjian/p/4564630.html)

 之前一直想弄theano，可是python不是很懂，在学习了一段时间之后开始安装theano。当然官网上的安装资料是全，可是也太繁琐了。这里介绍的是最简单，最方面的安装theano的方法。官网首页：<http://deeplearning.net/software/theano/>

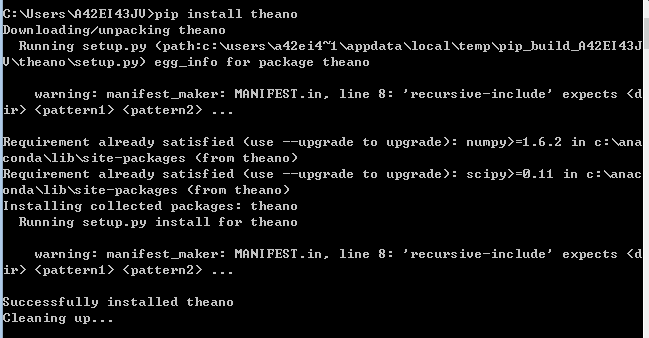
环境：win8.1\_64bit+ anaconda2.1.0\_64bit+CUDA6.5\_64bit+theano0.7。

1：下载Anaconda 2.1.0

https://repo.continuum.io/archive/.winzip/   ；该链接是anaconda官网的不同时间的版本。anaconda3表示python3.X，而anaconda表示2.X。这个正常安装就好，我是装在C:\Anaconda的，也就是C根目录下。无需重启（最好重启使得安装的path等路径生效）。

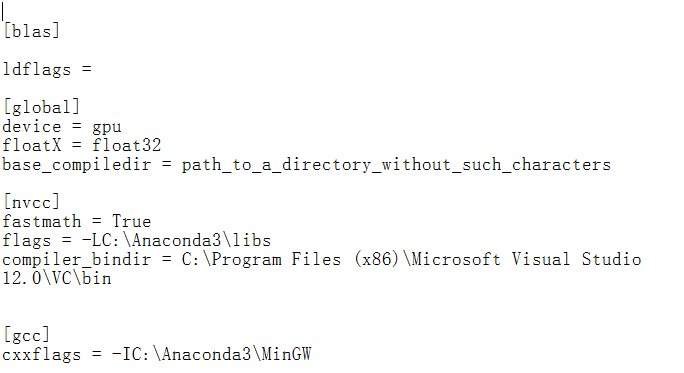
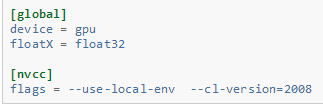
2：打开cmd，http://images0.cnblogs.com/blog2015/441382/201505/260938374127663.png，可以查看到有http://images0.cnblogs.com/blog2015/441382/201505/260939235531327.png，如果没这个，会在后续提示如“g++”找不到，无法编译的问题。

3：输入http://images0.cnblogs.com/blog2015/441382/201505/260941246939202.png，会自动在网上搜索theano0.7版本进行安装，会出现

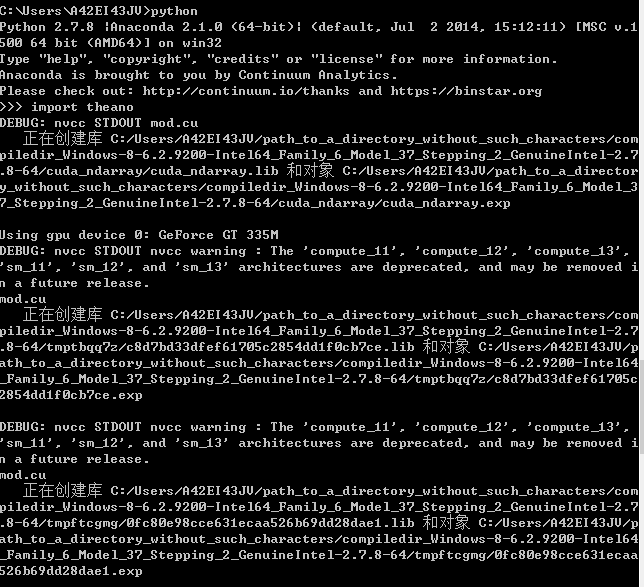
4：这时候其实theano应该算是安装好了，可是如果说只是用cpu，那么体现不出gpu的优势，可以在http://images0.cnblogs.com/blog2015/441382/201505/260946340847690.png这个路径下，也就是linux的/home路径，也就是windows的cmd命令运行的时候的路径



新建文件   .theanorc.txt，在里面写上

或官网的

5：输入python进入python环境，然后进行import theano，会出现：



按照之前找的资料，是不会出现这些东西的，为何我出现了，那就不知道。

6：随便建立个测试py文件，写上

[复制代码](javascript:void(0);)

复制代码

import numpy as np

import time

import theano

A = np.random.rand(1000,10000).astype(theano.config.floatX)

B = np.random.rand(10000,1000).astype(theano.config.floatX)

np\_start = time.time()

AB = A.dot(B)

np\_end = time.time()

X,Y = theano.tensor.matrices('XY')

mf = theano.function([X,Y],X.dot(Y))

t\_start = time.time()

tAB = mf(A,B)

t\_end = time.time()

print "NP time: %f[s], theano time: %f[s] (times should be close when run on CPU!)" %(

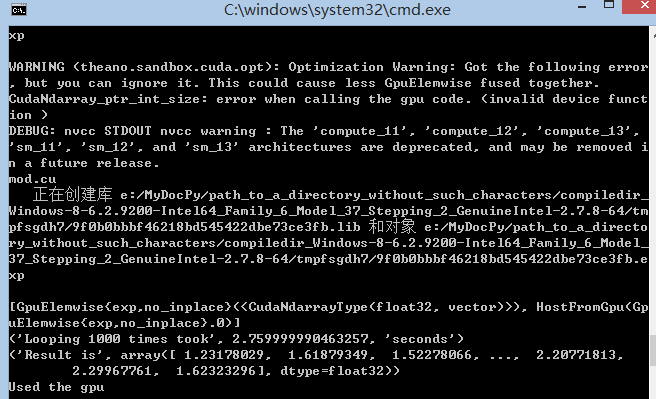
np\_end-np\_start, t\_end-t\_start)

print "Result difference: %f" % (np.abs(AB-tAB).max(), )

复制代码

[复制代码](javascript:void(0);)

不过注意这里的print不再是2.X的语句，而是需要使用迁移到3.X的函数形式，也就是使用print（）。运行后结果：



上图中最后的used the gpu 就是成功使用的证据，虽然上面的那一些出来的东西还暂时搞不懂，不过至少迈出了第一步。

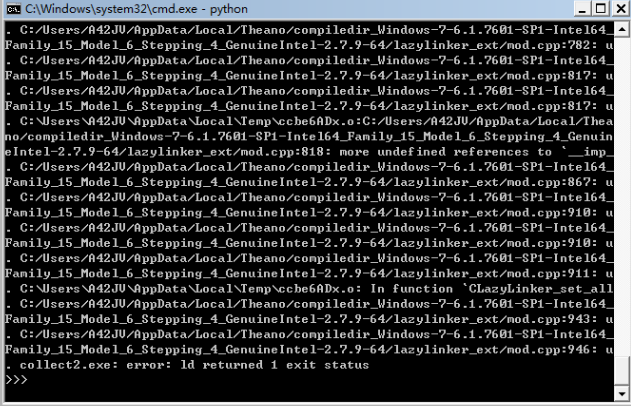
**注意事项：**

    1、为了安装的顺利，在安装anaconda和theano等软件的时候推荐路径中不要有空格，而且最好不要出现中文。

    2、首先暂时不论系统是多少位的，你的CUDA装的是32的，那么anaconda也对应下32位的，反之亦然。不过CUDA在7以上版本就开始独立一个文件不区分是否是32还是64位了，可是暂时没看到有用CUDA7来安装Theano的。

    3、虽然对于当前时间来说有更新的anaconda2.2.0的版本，但是还是推荐下这个，原因：

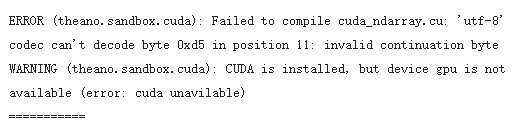
         a）不同版本中包含的东西不一样，这个版本的包含了mingw，而2.2.0版本的并不包含；虽然可以后续使用例如http://images0.cnblogs.com/blog2015/441382/201505/252239462094693.png[5]或者http://images0.cnblogs.com/blog2015/441382/201505/252240348819328.png,[5]来进行安装，可是后续还是会出问题如图：



该问题在[3]中的https://github.com/Theano/Theano/issues/2087里面有讨论，不过估计（猜的）是新的anaconda缺少mingw，而使用指令更新后得到的mingw不是原来的版本。

        b）theano对python3支持不是很好，所以不推荐下anaconda3系列的，因为theano是基于python2.4以上写的，这个在官网http://deeplearning.net/software/theano/install.html#install 中有说明http://images0.cnblogs.com/blog2015/441382/201505/260904133348743.png

如果安装python3的话，在后续import theano的时候会出现[4]中的http://comments.gmane.org/gmane.comp.mathematics.theano.user/6515的问题



（这个观点是按照字面意思猜的，不一定正确，不过该问题却已经解决）这是因为python2.X和python3.X中的默认编码选择已经不一样了，在3中默认是uniconde编码方式的。虽然在日常的代码编写中可以如http://www.cnblogs.com/blueel/archive/2012/08/21/2649590.html这样简单处理，可是对于这个问题，cuda\_ndarray.cu，首先进去看不到对应的字符串，而且感觉真正的原因不是这个。（https://github.com/Theano/Theano/issues/783，这个网页没看，不过对这个问题的理解有帮助）。

**未做：**

    1、如果想在VS下做图形化界面的开发，推荐[7]的对应部分。

**参考资料：**

[1] Theano 官网：http://deeplearning.net/software/theano/index.html

[2] Theano在github上的网址：https://github.com/Theano

[3] Theano在github上的问题讨论区：https://github.com/Theano/Theano/issues

[4] Theano的google团队问题讨论区：http://blog.gmane.org/gmane.comp.mathematics.theano.user

[5] Deep learning：Theano安装攻略：http://blog.163.com/yuyang\_tech/blog/static/216050083201469101518900/

[6] 采用anaconda2.1.0的灵感来源：http://blog.csdn.net/u010454729/article/details/43052505

[7] Win8.1\_64bit下搭建theano的博客：http://www.th7.cn/Program/Python/201407/228314.shtml

## [Theano2.1.1-基础知识之准备工作](http://www.cnblogs.com/shouhuxianjian/p/4564622.html)

来源：http://deeplearning.net/software/theano/tutorial/index.html#tutorial

    这里介绍的是使用theano的一些基础知识，虽然theano是用来dl 的，不过其中的一些函数还是挺有用的，可以用在ml 里面。

    在python或者ipython的交互模式下，可以按照如下形式来import theano：

>>> from theano import \*

    这里有几个你需要使用的符号都在theano的tensor这个子包中。让我们首先导入这个子包，并给它赋值一个新的变量名比如 T ：

>>> import theano.tensor as T

    如果这一步没报错，那么就可以准备接下来的教程了，不然还是回头去安装好theano吧（ [Installing Theano](http://deeplearning.net/software/theano/install.html#install))，[我的安装过程](http://blog.csdn.net/shouhuxianjian/article/details/46004159)。

    在整个教程中，记得有一个术语表（ [Glossary](http://deeplearning.net/software/theano/glossary.html#glossary) ，就在该博文下面这个目录的下面）可以用来作为索引和模块链接的。

**下面是 theano basic tutorial的目录（其中1、2、23节内容都在本页，其他的另成一页）：**

* 1、[**Python**](http://deeplearning.net/software/theano/tutorial/python.html)**简单教程**，下面是4个网页链接，其中有经典的练习和书籍
  + [Python](http://www.pythonchallenge.com/)的经典的挑战游戏，完成一个网页才能到达另一个网页
  + [Dive into Python](http://diveintopython.net/)
  + [Google Python Class](https://developers.google.com/edu/python/)
  + [Enthought Python course](https://training.enthought.com/?utm_source=academic&utm_medium=email&utm_campaign=EToD-Launch#/courses) (free for academics)
* **2、**[**NumPy**](http://deeplearning.net/software/theano/tutorial/numpy.html)**复习**
* **2.1机器学习的矩阵约定**
* 行表示水平方向，列表示竖直方向。每一行就是一个样本。所以inputs[10,5] 就是一个有着10个样本的矩阵，其中每个样本的维度为5。如果这是一个NN的输入，那么从输入到第一层隐藏层的权重就表示为size (5, #hid)的矩阵。

考虑这样一个数组:

[复制代码](javascript:void(0);)

>>> numpy.asarray([[1., 2], [3, 4], [5, 6]])

array([[ 1., 2.],

[ 3., 4.],

[ 5., 6.]])

>>> numpy.asarray([[1., 2], [3, 4], [5, 6]]).shape

(3, 2）

[复制代码](javascript:void(0);)

这是一个3\*2的矩阵，即有3行2列。

访问第3行（row#2）第1列(column #0)的元素：

>>> numpy.asarray([[1., 2], [3, 4], [5, 6]])[2, 0]

5.0

这里需要注意的是，我们是从左到右，从上到下读取的，所以一行是连续的，也就是3行2列。

### 2.2广播（Broadcasting）

Numpy 会在算术操作的时候对不同形状的数组进行广播。也就是说更小的那个数组（或者标量）会广播成对应的那个更大的数组，从而能够进行数学计算。下面就是一个广播的例子：

>>> a = numpy.asarray([1.0, 2.0, 3.0])

>>> b = 2.0

>>> a \* b

array([2., 4., 6.])

更小的数组b (实际上这里只是个标量, 可以看成是一个0d数组) 在这种情况下，在乘法运算中就会广播成相同的size。该技巧通常用在简化所写的表达式上。 更多有关广播的详细细节请查看[numpy user guide](http://docs.scipy.org/doc/numpy/user/basics.broadcasting.html).

* **3、**[**Baby Steps - Algebra**](http://deeplearning.net/software/theano/tutorial/adding.html)[**翻译**](http://blog.csdn.net/shouhuxianjian/article/details/46483113)
* **4、**[**More Examples**](http://deeplearning.net/software/theano/tutorial/examples.html)[**翻译**](http://blog.csdn.net/shouhuxianjian/article/details/46494025)
* **5、**[**Graph Structures**](http://deeplearning.net/software/theano/tutorial/symbolic_graphs.html)[**翻译**](http://blog.csdn.net/shouhuxianjian/article/details/46509271)
* **6、**[**Printing/Drawing Theano graphs**](http://deeplearning.net/software/theano/tutorial/printing_drawing.html)[**翻译**](http://blog.csdn.net/shouhuxianjian/article/details/46516073)
* **7、**[**Derivatives in Theano**](http://deeplearning.net/software/theano/tutorial/gradients.html)[**翻译**](http://blog.csdn.net/shouhuxianjian/article/details/46517143)
* **8、**[**Configuration Settings and Compiling Modes**](http://deeplearning.net/software/theano/tutorial/modes.html)[**翻译**](http://blog.csdn.net/shouhuxianjian/article/details/46520729)
* **9、**[**Loading and Saving**](http://deeplearning.net/software/theano/tutorial/loading_and_saving.html)[**翻译**](http://blog.csdn.net/shouhuxianjian/article/details/46523711)
* **10、**[**Conditions**](http://deeplearning.net/software/theano/tutorial/conditions.html)[**翻译**](http://blog.csdn.net/shouhuxianjian/article/details/46524871)
* **11、**[**Loop**](http://deeplearning.net/software/theano/tutorial/loop.html)[**翻译**](http://blog.csdn.net/shouhuxianjian/article/details/46525035)
* **12、**[**Sparse**](http://deeplearning.net/software/theano/tutorial/sparse.html)[**翻译**](http://blog.csdn.net/shouhuxianjian/article/details/46531411)
* **13、**[**Using the GPU**](http://deeplearning.net/software/theano/tutorial/using_gpu.html)[**翻译**](http://blog.csdn.net/shouhuxianjian/article/details/46537463)
* **14、**[**PyCUDA/CUDAMat/Gnumpy compatibility**](http://deeplearning.net/software/theano/tutorial/gpu_data_convert.html)[**翻译**](http://blog.csdn.net/shouhuxianjian/article/details/46547067)
* **15、**[**Understanding Memory Aliasing for Speed and Correctness**](http://deeplearning.net/software/theano/tutorial/aliasing.html)[**翻译**](http://blog.csdn.net/shouhuxianjian/article/details/46548897)
* **16、**[**How Shape Information is Handled by Theano**](http://deeplearning.net/software/theano/tutorial/shape_info.html)[**翻译**](http://blog.csdn.net/shouhuxianjian/article/details/46558639)
* **17、**[**Debugging Theano: FAQ and Troubleshooting**](http://deeplearning.net/software/theano/tutorial/debug_faq.html)[**翻译**](http://blog.csdn.net/shouhuxianjian/article/details/46561123)
* **18、**[**Profiling Theano function**](http://deeplearning.net/software/theano/tutorial/profiling.html)[**翻译**](http://blog.csdn.net/shouhuxianjian/article/details/46566011)
* **19、**[**Extending Theano**](http://deeplearning.net/software/theano/tutorial/extending_theano.html)[**翻译**](http://blog.csdn.net/shouhuxianjian/article/details/46571907)
* **20、**[**Extending Theano with a C Op**](http://deeplearning.net/software/theano/tutorial/extending_theano_c.html)
* **21、**[**Python Memory Management**](http://deeplearning.net/software/theano/tutorial/python-memory-management.html)
* **22、**[**Multi cores support in Theano**](http://deeplearning.net/software/theano/tutorial/multi_cores.html)[**翻译**](http://blog.csdn.net/shouhuxianjian/article/details/46591551)
* **23、**[**Frequently Asked Questions**](http://deeplearning.net/software/theano/tutorial/faq_tutorial.html)
* **23.1如何更新权重的子集?**

如果你只想要更新一个权重矩阵的子集（例如某些行，某些列），这种情况在每次迭代的前向传播过程中会遇到。那么cost函数应该被定义成只依赖在这次迭代中当前权重子集的方式才行。

例如，如果你想要学习一个查找表，例如，在处理[词向量](http://blog.sina.com.cn/s/blog_584a006e0101rjlm.html)（word embedding）的时候，每一行的权重向量用来表示模型从一个单词学到的embedding，在每次迭代中，那些唯一需要更新的行都是在前馈传播中包含着embedding的。这里介绍下theano 函数应该如何来写：

对查找表定义一个共享变量

>>> lookup\_table = theano.shared(matrix\_ndarray).

通过传递所需要行或者列的整数索引向量来得到这个表的一个子集 (某些行或者某些列) .

>>> subset = lookup\_table[vector\_of\_indices]

从现在开始，使用 ‘subset’，而不会再调用 lookup\_table[vector\_of\_indices] 了。这会造成梯度上的问题，因为这会生成新的变量

定义cost只依赖于subset而不是整个lookup\_table：

>>> cost = something that depends on subset

>>> g = theano.grad(cost, subset)

有两种方式来更新这些参数：使用inc\_subtensor 或者set\_subtensor。推荐使用 inc\_subtensor。一些theano是在这两者之间通过转换来进行优化的，不过不是所有情况下都是如此：

>>> updates = inc\_subtensor(subset, g\*lr)

OR

>>> updates = set\_subtensor(subset, subset + g\*lr)

如果你不使用inc\_subtensor或set\_subtensor与其他类型的索引，那么我们只需要掌握这里的情况就可以了。

定义这个theano函数

>>> f=theano.function(..., updates=updates)

注意到你可以计算cost函数关于整个lookup\_table的梯度, 在前馈传播过程中选取的行中会有非0行的梯度。 如果你使用梯度下降来进行更新这些参数，那么就不会有问题，除了不必要的计算，例如，你可能会使用许多梯度为0的行来更新这个查找表的参数。然而，如果你想要使用一个不同的优化方法比如rmsprop 或者 Hessian-Free 优化，那么就会有问题。在 rmsprop 中，你需要通过平方根除以当前的梯度（whose square root you divide the current gradient，不知道翻译的对不对）来逐部分的逐步重新调整，从而保持一个指数衰减平方梯度（exponentially decaying squared gradient）。如果对应一个罕见单词的查找表行的梯度经常出现0，那么对于这一行来说该梯度的平方将倾向于0，因为该行的衰减倾向于为0；使用Hessian-Free的话，你会得到许多的0行和0列，甚至它们中有些还是不可逆的。通常来说，只计算梯度关于在前馈传播中实际用到的查找表的行会更好。

### Glossary（术语表）

Apply

apply的实例表示从一些输入变量（[Variable](http://deeplearning.net/software/theano/glossary.html#term-variable) ）到生成一些输出变量（ [Variable](http://deeplearning.net/software/theano/glossary.html#term-variable) ）的操作( [Op](http://deeplearning.net/software/theano/glossary.html#term-op) ) 。就像是数学形式上的函数在输入的参数上的应用.

Broadcasting

广播这个机制允许有着不同数量维度的张量能够应用在逐元素处理的情形下。它通过在缺失的维度的方向上将更小的张量进行复制来实现的。

更详细的，可以看看[Broadcasting in Theano vs. Numpy](http://deeplearning.net/software/theano/library/tensor/basic.html#libdoc-tensor-broadcastable), 和 \* [SciPy documentation about numpy’s broadcasting](http://www.scipy.org/EricsBroadcastingDoc) \* [OnLamp article about numpy’s broadcasting](http://www.onlamp.com/pub/a/python/2000/09/27/numerically.html)

Constant

一个有着不可变值的变量。例如，当你输入如下：

>>> x = tensor.ivector()

>>> y = x + 3

那么在graph中就会创建一个常量3。

See also: gof.Constant

Elementwise

两个张量变量 M 和 N的一个逐元素操作 f 的形式如下：

f(M, N)[i, j] == f(M[i, j], N[i, j])

换句话说，输入矩阵中的每个元素是和其他矩阵对应位置上的元素相结合的。 当元素的坐标[i, j] 没有对应的时候，元素之间是没有依赖关系的，所以逐元素操作就像是在几个维度上的一个标量操作。 在不同维度张量上的逐元素操作可以通过 [broadcasting](http://deeplearning.net/software/theano/glossary.html#term-broadcasting) （广播）更小的维度来实现。

Expression

见 [Apply](http://deeplearning.net/software/theano/glossary.html#term-apply)

Expression Graph

[Variable](http://deeplearning.net/software/theano/glossary.html#term-variable) 和 用来表示变量之间的符号函数关系的[Apply](http://deeplearning.net/software/theano/glossary.html#term-apply) 节点相连可以构成一个有向，无环集合。你可以通过定义表达式图来使用theano，然后用 [theano.function](http://deeplearning.net/software/theano/glossary.html#term-theano-function)来对他们进行编译。

See also [Variable](http://deeplearning.net/software/theano/glossary.html#term-variable), [Op](http://deeplearning.net/software/theano/glossary.html#term-op), [Apply](http://deeplearning.net/software/theano/glossary.html#term-apply), 和 [Type](http://deeplearning.net/software/theano/glossary.html#term-type), 或者更多的可以看看 [Graph Structures](http://deeplearning.net/software/theano/tutorial/symbolic_graphs.html#tutorial-graphstructures).

Destructive

当计算一个或多个输入需要被重写要不然就会失效，那么该操作 [Op](http://deeplearning.net/software/theano/glossary.html#term-op) （对于某个具体的input[s]）就是破坏性的。例如，[inplace](http://deeplearning.net/software/theano/glossary.html#term-inplace) 操作就是破坏性的。不过破坏性的操作有时候比非破坏性的操作快很多。theano建议用户不要将破坏性的操作放入graph中，然后给  [theano.function](http://deeplearning.net/software/theano/glossary.html#term-theano-function) but 不过却可以可以通过对优化插入破坏性的操作来优化。

破坏性的操作可以通过 destroy\_map 的操作属性来指明。 (See gof.Op.

Graph

见 [expression graph](http://deeplearning.net/software/theano/glossary.html#term-expression-graph)

Inplace

Inplace计算是会带有破坏它们输入的副作用的。例如，如果你需要对一个矩阵进行迭代，其中的操作是将每个元素乘以2，那么这就是一个inplace操作，因为当你完成的时候，原始的输入已经被重写了。这说明inplace 计算是破坏性的 [destructive](http://deeplearning.net/software/theano/glossary.html#term-destructive), 默认情况下它们只能被用来作为优化的插入，而不是用户编写的代码。

Linker

[Mode](http://deeplearning.net/software/theano/glossary.html#term-mode) 函数的一部分，一个用来负责“运行”编译后函数的对象。在其他环境下，linker 决定着计算是用C 还是 Python 的代码。

Mode

一个提供了 [optimizer](http://deeplearning.net/software/theano/glossary.html#term-optimizer) 和 [linker](http://deeplearning.net/software/theano/glossary.html#term-linker) 的对象，会被传送给 [theano.function](http://deeplearning.net/software/theano/glossary.html#term-theano-function). 它可以将一个表达式图如何转换成一个可调用的对象进行参数化

Op

 一个 [Apply](http://deeplearning.net/software/theano/glossary.html#term-apply) 的 .op和它的符号输入一起决定着在运行的时候，哪种计算方式会被使用。在theano中，例如加法 (T.add) 和索引 x[i] 这样的数学上的函数都是操作。库文档中大部分都是用来描述不同的操作的，不过你可以自行添加其他的操作。

See also [Variable](http://deeplearning.net/software/theano/glossary.html#term-variable), [Type](http://deeplearning.net/software/theano/glossary.html#term-type), 和 [Apply](http://deeplearning.net/software/theano/glossary.html#term-apply), 更详细的可以看 [Graph Structures](http://deeplearning.net/software/theano/tutorial/symbolic_graphs.html#tutorial-graphstructures).

Optimizer

[Optimizer](http://deeplearning.net/software/theano/extending/optimization.html#Optimizer),的一个实例，可以提供 [optimization](http://deeplearning.net/software/theano/glossary.html#term-optimization) (或者 optimizations)。

Optimization

[optimizer](http://deeplearning.net/software/theano/glossary.html#term-optimizer) 应用的 [graph](http://deeplearning.net/software/theano/glossary.html#term-graph) 变换发生在[theano.function](http://deeplearning.net/software/theano/glossary.html#term-theano-function) 对 [graph](http://deeplearning.net/software/theano/glossary.html#term-graph)的编译的时候。

Pure

当没有发生破坏性的副作用的时候操作 [Op](http://deeplearning.net/software/theano/glossary.html#term-op) 是纯粹的。

Storage

T用来存储变量值的内存部分。在大多数情况下，存储是与编译后的函数内部相关的，不过在某些情况下 (例如 [constant](http://deeplearning.net/software/theano/glossary.html#term-constant) 和 [shared variable](http://deeplearning.net/software/theano/glossary.html#term-shared-variable) 的时候，存储就不是内部相关了。

Shared Variable

一个可以在不同的函数之间共享的变量（ [Variable](http://deeplearning.net/software/theano/glossary.html#term-variable) ） 。可以参考共享和[theano.function](http://deeplearning.net/software/theano/library/compile/function.html#function.function).

theano.function

将符号表达式图编译成可调用的对象的theano的接口。可以参考 [function.function()](http://deeplearning.net/software/theano/library/compile/function.html#function.function).

Type

[Variable](http://deeplearning.net/software/theano/glossary.html#term-variable)  .type 用来表示哪一种值会在编译后的graph中被计算。继承自**Type**的实例，用来作为一个变量 [Variable](http://deeplearning.net/software/theano/glossary.html#term-variable).的.type属性。

See also [Variable](http://deeplearning.net/software/theano/glossary.html#term-variable), [Op](http://deeplearning.net/software/theano/glossary.html#term-op), 和 [Apply](http://deeplearning.net/software/theano/glossary.html#term-apply), 更多可以看看[Graph Structures](http://deeplearning.net/software/theano/tutorial/symbolic_graphs.html#tutorial-graphstructures).

Variable

这是你使用theano的时候主要用到的数据结构。例如：

>>> x = theano.tensor.ivector()

>>> y = -x\*\*2

x 和 y 都是 Variables, 即，是Variable 这个类的实例。

See also [Type](http://deeplearning.net/software/theano/glossary.html#term-type), [Op](http://deeplearning.net/software/theano/glossary.html#term-op), and [Apply](http://deeplearning.net/software/theano/glossary.html#term-apply), 或者更详细的可以看看 [Graph Structures](http://deeplearning.net/software/theano/tutorial/symbolic_graphs.html#tutorial-graphstructures).

View

一些张量操作 (例如 Subtensor 和 Transpose) 可以通过简单的对输入重新索引来保持在常量时间内完成。从apply的实例得到输出的这样的操作被称为 Views 是因为它们的存储可能会被其他变量（apply的输入）的存储别名了. 对于theano来说，知道哪些变量是其他变量的views是很重要的，因为这样就可以引入正确的 [Destructive](http://deeplearning.net/software/theano/glossary.html#term-destructive) 操作.

View 操作是通过一个 view\_map 操作属性来表示的。 (See gof.Op.

[**Theano2.1.2-基础知识之第一步：代数**](http://www.cnblogs.com/shouhuxianjian/p/4590234.html)

来自：http://deeplearning.net/software/theano/tutorial/adding.html

Baby Steps - Algebra

**一、两个标量相加**

    在学习theano的开始，首先来感受下它是如何工作的，让我们先来定义一个简单的相加函数：

>>> import theano.tensor as T

>>> from theano import function

>>> x = T.dscalar('x')

>>> y = T.dscalar('y')

>>> z = x + y

>>> f = function([x, y], z)

    现在，我们可以使用之前创建好的函数，然后使用了：

>>> f(2, 3)

array(5.0)

>>> f(16.3, 12.1)

array(28.4)

    这里是将上面部分进行分步介绍。首先定义两个符号 (变量)来表示你想要相加的数。注意到从现在开始，我们将会使用变量（*Variable*)来表示“符号”(换句话说，也就是 *x*, *y*, *z* 都是变量对象)。函数 f 的输出就是一个有着0维的 numpy.ndarray 。

    如果你想要输入一个整数，你可能会发现在执行function的时候有轻微的延时。这是因为需要将函数 f 编译成c 代码。

**第一步**

>>> x = T.dscalar('x')

>>> y = T.dscalar('y')

    在theano中，所有的符号都必须手动输入。具体来说，T.dscalar 就是我们分配给“doubles类型（d）的0维数组（scalar）”类型。这是一个theano类型 [*Type*](http://deeplearning.net/software/theano/extending/graphstructures.html#type).

  dscalar 不是一个类，所以 *x* 和 *y* 都不是 dscalar的实例。他们是 **TensorVariable**的实例 。正如下面看到的：

>>> type(x)

<class 'theano.tensor.basic.TensorVariable'>

>>> x.type

TensorType(float64, scalar)

>>> T.dscalar

TensorType(float64, scalar)

>>> x.type is T.dscalar

True

    通过一个字符串参数来调用 T.dscalar ，可以创建一个变量来表示一个给定名字下的一个浮点标量。如果你不提供参数，该符号将会是未命名的（unnamed）。虽然名字是不是必须的，可是却有助于调试。

    更多的可以看看theano的内部结构。同样可以看看 [*Graph Structures*](http://deeplearning.net/software/theano/extending/graphstructures.html#graphstructures).

**第二步**

    第二步就是结合 *x* 与*y* 到它们的和*z中*

>>> z = x + y

    z 是另一个变量用来表示 x 与 y的和. 你可以使用 [*pp*](http://deeplearning.net/software/theano/library/printing.html#libdoc-printing) 函数来友好的打印出与z 关联的计算结果。

>>> from theano import pp

>>> print pp(z)

(x + y)

**第三步**

    最后一步就是创建一个函数，将x 与 y  作为输入，然后通过z  作为输出：

>>> f = function([x, y], z)

    [**function**](http://deeplearning.net/software/theano/library/compile/function.html#function.function) 的第一个参数是变量的列表，用来作为输入提供给函数。第二个参数是一个单一的变量或者一个变量列表。在其他情况下，第二个变参数就是我们想要的输出。然后f 就可以被和普通的python函数一样使用了。  
note：作为一个快捷方式，你可以跳过第三步，值使用一个变量的**eval()** 方法。 **eval()** 方法没有 [**function()**](http://deeplearning.net/software/theano/library/compile/function.html#module-function) 这么灵活，不过却能做教程中的任何事情。这是你不需要inport [**function()**](http://deeplearning.net/software/theano/library/compile/function.html#module-function) 的额外的福利. 下面就是 **eval()** 工作的过程：

>>> import theano.tensor as T

>>> x = T.dscalar('x')

>>> y = T.dscalar('y')

>>> z = x + y

>>> z.eval({x : 16.3, y : 12.1})

array(28.4)

我们给 **eval()**传递一个词典，将符号theano变量映射成值从而来取代它们，然后对表达式返回数值。

**eval()** 在第一次调用的时候可能会比较慢，因为它需要调用 [**function()**](http://deeplearning.net/software/theano/library/compile/function.html#module-function) 来计算场景后面的表达式。之后对 **eval()** 以同样的变量上的调用将会很快，因为变量已经将编译后的函数进行缓存了。

**二、两个矩阵相加**

    你可能已经猜到如何来做了确实，这里与之前唯一的区别在于你需要实例化x 与y ，然后使用矩阵类型：

>>> x = T.dmatrix('x')

>>> y = T.dmatrix('y')

>>> z = x + y

>>> f = function([x, y], z)

  dmatrix 是doubles类型的矩阵，然后我们就能在2D数组上使用新的函数了：

>>> f([[1, 2], [3, 4]], [[10, 20], [30, 40]])

array([[ 11., 22.],

[ 33., 44.]])

    该变量是一个 NumPy 数组。我们同样可以直接使用NumPy 数组作为输入：

>>> import numpy

>>> f(numpy.array([[1, 2], [3, 4]]), numpy.array([[10, 20], [30, 40]]))

array([[ 11., 22.],

[ 33., 44.]])

    计算矩阵加标量、矩阵加向量、向量加标量都是可以的。这些能够得以实现得归功于[*broadcasting*](http://deeplearning.net/software/theano/library/tensor/basic.html#libdoc-tensor-broadcastable).

    下面的类型都是可用的：

* **byte**: bscalar, bvector, bmatrix, brow, bcol, btensor3, btensor4
* **16-bit integers**: wscalar, wvector, wmatrix, wrow, wcol, wtensor3, wtensor4
* **32-bit integers**: iscalar, ivector, imatrix, irow, icol, itensor3, itensor4
* **64-bit integers**: lscalar, lvector, lmatrix, lrow, lcol, ltensor3, ltensor4
* **float**: fscalar, fvector, fmatrix, frow, fcol, ftensor3, ftensor4
* **double**: dscalar, dvector, dmatrix, drow, dcol, dtensor3, dtensor4
* **complex**: cscalar, cvector, cmatrix, crow, ccol, ctensor3, ctensor4

     上面的列表不够详尽，针对使用Numpy数组来兼容所有类型的更详细文档可以看: [*tensor creation*](http://deeplearning.net/software/theano/library/tensor/basic.html#libdoc-tensor-creation).

note：你，用户（不是系统架构者）不得不选择你的程序为32- 或者 64-bit 的整数 (i 前缀 vs. l 前缀) 和浮点数 (f 前缀 vs. d 前缀).

**三、练习**

import theano

a = theano.tensor.vector() # declare variable

out = a + a \*\* 10 # build symbolic expression

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

print f([0, 1, 2]) # prints `array([0, 2, 1026])`

 修改并计算这个表达式的结果: a \*\* 2 + b \*\* 2 + 2 \* a \* b.

下面就是答案：

#!/usr/bin/env python

# Theano tutorial

# Solution to Exercise in section 'Baby Steps - Algebra'

from \_\_future\_\_ import print\_function

import theano

a = theano.tensor.vector() # declare variable

b = theano.tensor.vector() # declare variable

out = a \*\* 2 + b \*\* 2 + 2 \* a \* b # build symbolic expression

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

print(f([1, 2], [4, 5])) # prints [ 25. 49.]

**参考资料：**

[1] 官网：http://deeplearning.net/software/theano/tutorial/adding.html

## [Theano2.1.3-基础知识之更多的例子](http://www.cnblogs.com/shouhuxianjian/p/4594517.html)

来自：http://deeplearning.net/software/theano/tutorial/examples.html

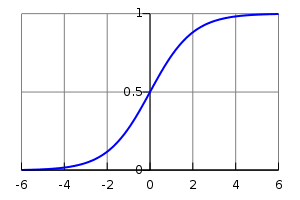
More Examples

    现在，是时候开始系统的熟悉theano的基础对象和操作了，可以通过浏览库的部分来详细的了解 [Basic Tensor Functionality](http://deeplearning.net/software/theano/library/tensor/basic.html#libdoc-basic-tensor). 随着这个教程的深入，你可以逐渐的让自己熟悉库的其他相关的部分和文档入口页面的其他相关的主题了。

一、Logistic 函数

     这是一个简单的例子，虽然会比两个数值相加要难一些。假设你想要计算一个逻辑曲线，首先得到一个如下的式子：

s(x) = \frac{1}{1 + e^{-x}}



上图是一个逻辑函数，x轴表示x值，y轴表示s（x）值。

你需要在doubles矩阵上逐元素（ [elementwise](http://deeplearning.net/software/theano/library/tensor/basic.html#libdoc-tensor-elementwise) ）的计算这个函数，也就是说你是想要在矩阵的每个独立的元素上都使用该函数。代码如下：

**[python]** [view plaincopy](http://blog.csdn.net/shouhuxianjian/article/details/46494025)

1. >>> x = T.dmatrix('x')
2. >>> s = 1 / (1 + T.exp(-x))
3. >>> logistic = function([x], s)
4. >>> logistic([[0, 1], [-1, -2]])
5. array([[ 0.5       ,  0.73105858],
6. [ 0.26894142,  0.11920292]])

    需要逐元素计算是因为它的操作：除法、加法、指数和减法，都是逐元素的操作。在该情况下也是：

s(x) = \frac{1}{1 + e^{-x}} = \frac{1 + \tanh(x/2)}{2}

我们可以验证从这个可代替的式子上得到的结果是一样的：

1. >>> s2 = (1 + T.tanh(x / 2)) / 2
2. >>> logistic2 = function([x], s2)
3. >>> logistic2([[0, 1], [-1, -2]])
4. array([[ 0.5       ,  0.73105858],
5. [ 0.26894142,  0.11920292]])

# 二、在同一时间对多个操作进行计算

    Theano支持函数有着多于一个的输出。例如，我们可以在同一时刻计算两个矩阵a和b 之间的逐元素（[elementwise](http://deeplearning.net/software/theano/library/tensor/basic.html#libdoc-tensor-elementwise) ）的差，绝对值的差,平方值的差：

1. >>> a, b = T.dmatrices('a', 'b')
2. >>> diff = a - b
3. >>> abs\_diff = abs(diff)
4. >>> diff\_squared = diff\*\*2
5. >>> f = function([a, b], [diff, abs\_diff, diff\_squared])

note：dmatrices 生成提供的名字一样数量的输出。这是一个用来分配符号变量的快捷方式，在本教程中会经常用到。

    当我们使用函数f 时，它返回三个变量（输出的时候会为了更好的可读性而被重新格式）：

1. >>> f([[1, 1], [1, 1]], [[0, 1], [2, 3]])
2. [array([[ 1.,  0.],
3. [-1., -2.]]),
4. array([[ 1.,  0.],
5. [ 1.,  2.]]),
6. array([[ 1.,  0.],
7. [ 1.,  4.]])]

# 三、对参数设置默认值

    假设你想要定义一个相加两个数的函数，如果你定义完之后，在调用的时候，只提供了一个参数，那么另一个输入可以假设默认为1，可以如下所示：

1. >>> **from theano import Param**
2. >>> x, y = T.dscalars('x', 'y')
3. >>> z = x + y
4. >>> f = function([x, Param(y, default=1)], z)
5. >>> f(33)
6. array(34.0)
7. >>> f(33, 2)
8. array(35.0)

    使用的 [Param](http://deeplearning.net/software/theano/library/compile/io.html#function-inputs) 参数允许你指定你函数的参数有着更详细的值。这里我们通过创建一个Param实例来将y设置其默认值为1。

    有着默认值的输入必须在没有默认值的输入的后面（和python的函数一样的顺序）可以对多个输入进行设置莫仍只。这些蚕食可以通过位置或者名称来进行设定，就像标准的python中一样：

1. >>> x, y, w = T.dscalars('x', 'y', 'w')
2. >>> z = (x + y) \* w
3. >>> f = function([x, Param(y, default=1), Param(w, default=2, name='w\_by\_name')], z)
4. >>> f(33)
5. array(68.0)
6. >>> f(33, 2)
7. array(70.0)
8. >>> f(33, 0, 1)
9. array(33.0)
10. >>> f(33, w\_by\_name=1)
11. array(34.0)
12. >>> f(33, w\_by\_name=1, y=0)
13. array(33.0)

note：Param 不知道作为参数传入的局部变量y 和w 的名称。这些符号变量对象都有name属性（和上面例子一样通过dscalars来设置），这些是我们构建的函数中的关键参数的名称。这就是 Param(y, default=1)中的工作机制。在Param(w, default=2, name='w\_by\_name')的情况下 ，我们用在这个函数中使用过的名字来覆盖符号变量的名字属性。  
    你可以看看库中的 [Function](http://deeplearning.net/software/theano/library/compile/function.html#usingfunction) 来更详细的了解。

# 四、使用共享变量

      同样的也可以让函数有一个内部状态。例如，我们想要在开始就设置一个累加器，先初始化为0。那么，在每次的函数调用， 该状态就会被函数的参数递增的。首先定义一个accumulator 函数。然后将参数增加到内部状态上，然后返回增加之前的状态值。

    该代码引入了一些新的概念。 shared 函数构建所谓的 [shared variables](http://deeplearning.net/software/theano/library/compile/shared.html#libdoc-compile-shared)。这些都是混合符号和非符号变量，他们的值可以在多个函数中共享，就像是由dmatrices(...)返回的对象一样，不过他们同样有着一个内部值，这个值是通过这个在所有函数中使用的符号变量定义的。被称作共享变量是因为它的值在许多函数之间共享的。该值可以被 .get\_value() 和 .set\_value() 方法所访问和修改。

    该代码中另一个新事物就是function. updates的参数 updates 必须被以(shared-variable, new expression)这种对形式的列表所赋值。它同样可以是一个字典，其中的键是共享变量而值是新表达式。。不管怎么说，它表示“不论什么时候运行，它会将.value的每个共享变量替换成对应的表达式的结果” 。也就是说，我们的累加器会用状态state的和以及递增数来替换状态state的值。

1. >>> state.get\_value()
2. array(0)
3. >>> accumulator(1)
4. array(0)
5. >>> state.get\_value()
6. array(1)
7. >>> accumulator(300)
8. array(1)
9. >>> state.get\_value()
10. array(301)

可以使用 .set\_value() 方法来重置状态state:

1. >>> state.set\_value(-1)
2. >>> accumulator(3)
3. array(-1)
4. >>> state.get\_value()
5. array(2)

    正如上面说的，你可以定义超过一个函数来使用相同的共享变量。这些函数都能够更新这个值。

1. >>> decrementor = function([inc], state, updates=[(state, state-inc)])
2. >>> decrementor(2)
3. array(2)
4. >>> state.get\_value()
5. array(0)

    你也许会惊讶为什么这个更新机制会存在。你总可以通过返回一个新的表达式来得到一个相似的结果，然后在NumPy里面使用它们。该更新机制是一个语法上的方便，不过在这里主要是因为效率问题。对共享变量的更新有时候可以使用in-place算法更快的完成(了例如： low-rank矩阵更新).。同样的，theano有着更多有关在哪和如何来分配共享权重的函数，这些都是在需要使用在 [GPU](http://deeplearning.net/software/theano/tutorial/using_gpu.html#using-gpu)上很重要的组成部分.

  有时候你想要使用一个共享变量来表示一些公式，却不想要使用它们的值。在这种情况下，你可以使用function函数的givens的参数，这个用来代替这种情况下graph中的特定的节点。

1. >>> fn\_of\_state = state \* 2 + inc
2. >>> # The type of foo must match the shared variable we are replacing
3. >>> # with the ``givens``
4. >>> foo = T.scalar(dtype=state.dtype)
5. >>> skip\_shared = function([inc, foo], fn\_of\_state,
6. givens=[(state, foo)])
7. >>> skip\_shared(1, 3)  # we're using 3 for the state, not state.value
8. array(7)
9. >>> state.get\_value()  # old state still there, but we didn't use it
10. array(0)

     givens 参数可以用来代替任何符号变量，不只是共享变量。你还可以用来代替常量、表达式。不过要注意，不要让由givens替换的表达式之间有着相互依赖关系，替换的顺序是没法保证的，所以替换之后是有可能以任意顺序来执行的。

    在实际中，有关使用givens的一个好的方法就是替换公式的任何部分的时候使用的是不同的表达式，只不过该表达式有着相同shape和dtype的张量而已。

note：Theano 共享变量的广播模式默认情况下对于每个维度来说都是False。共享变量的size可以随着时间变化，所以我们没法使用shape来找到可广播的模式。如果你想要一个不同的模式，只要将它像参数一样传递 theano.shared(..., broadcastable=(True, False))。

# 五、使用随机数

      因为在theano中，你首先会将任何事情进行符号化，然后编译这个表达式来得到函数，然后使用伪随机数不是和Numpy中一样简单的，当然也不会太复杂。

    将随机放入theano的计算中就是将随机变量放到你的graph中。theano将会对每个这样的变量分配一个 NumPy RandomStream 对象 (一个随机生成器) ，然后必要的时候提取出来。我们称这类随机数序列为a random stream. 随机流的核心也是共享变量，所以对共享变量的观察在这里也是一样的。theano的随机对象的定义和实现在 [RandomStreams](http://deeplearning.net/software/theano/library/tensor/shared_randomstreams.html#libdoc-tensor-shared-randomstreams) 更低的版本，也就是其父类[RandomStreamsBase](http://deeplearning.net/software/theano/library/tensor/raw_random.html#libdoc-tensor-raw-random).

### 5.1 一个简单的例子

    这里是一个简短的例子，构建代码为：

1. **from theano.tensor.shared\_randomstreams import RandomStreams**
2. **from theano import function**
3. srng = RandomStreams(seed=234)
4. rv\_u = srng.uniform((2,2))
5. rv\_n = srng.normal((2,2))
6. f = function([], rv\_u)
7. g = function([], rv\_n, no\_default\_updates=True)    #Not updating rv\_n.rng
8. nearly\_zeros = function([], rv\_u + rv\_u - 2 \* rv\_u)

    这里 ‘rv\_u’ 表示一个从均匀分布中提取的2\*2的矩阵的随机流。同样的， ‘rv\_n’ 表示一个从正太分布中提取的2\*2矩阵的一个随机流。该分布实现和定义都在 RandomStreams 中，其父类为 [raw\_random](http://deeplearning.net/software/theano/library/tensor/raw_random.html#libdoc-tensor-raw-random). 他们只在CPU上工作。 [Other Implementations](http://deeplearning.net/software/theano/tutorial/examples.html#other-implementations) 上面有GPU版本的。

    现在让我们来使用这些对象。如果我们调用 f()，我们就得到了均匀随机数。 随机数生成器的内部状态是自动进行更新的，所以我们在每个时间上得到的是不同的随机数：

1. >>> f\_val0 = f()
2. >>> f\_val1 = f()  #different numbers from f\_val0

    当我们增加额外的参数 no\_default\_updates=True 到 function (as in g)，那么随机数生成器状态不会受到返回函数调用的影响。所以，例如，调用g 函数多次，而返回的是同样的数值：

1. >>> g\_val0 = g()  # different numbers from f\_val0 and f\_val1
2. >>> g\_val1 = g()  # same numbers as g\_val0!

    一个重要的备注是随机变量在任何单一函数执行中最多被提取一次。所以，即使 rv\_u 随机变量在输出表达式中出现了三次，nearly\_zero函数可以保证返回的值可以逼近0 (除了舍入导致的错误)：

1. >>> nearly\_zeros = function([], rv\_u + rv\_u - 2 \* rv\_u)

### 5.2 Seeding Streams

    随机变量可以被独立或集体的被传入种子。你可以只对随机变量通过seeding或者使用.rng.set\_value()的.rng 属性来指定传入种子：

1. >>> rng\_val = rv\_u.rng.get\_value(borrow=True)   # Get the rng for rv\_u
2. >>> rng\_val.seed(89234)                         # seeds the generator
3. >>> rv\_u.rng.set\_value(rng\_val, borrow=True)    # Assign back seeded rng

   你同样可以对所有的随机变量通过**RandomStreams**对象的seed方法来分配种子。该种子将会被用来传递给一个临时随机数生成器，然后对每个随机变量生成种子。

1. >>> srng.seed(902340)  # seeds rv\_u and rv\_n with different seeds each

### 5.3 在函数之间共享流

    和通常的共享变量一样，用于随机变量的随机数生成器都是函数中常见的。所以我们的nearly\_zeros 函数将会使用上面介绍的函数 f 来更新生成器的状态。例如：

1. >>> state\_after\_v0 = rv\_u.rng.get\_value().get\_state()
2. >>> nearly\_zeros()       # this affects rv\_u's generator
3. >>> v1 = f()
4. >>> rng = rv\_u.rng.get\_value(borrow=True)
5. >>> rng.set\_state(state\_after\_v0)
6. >>> rv\_u.rng.set\_value(rng, borrow=True)
7. >>> v2 = f()             # v2 != v1
8. >>> v3 = f()             # v3 == v1

### 5.4 在theano graphs之间复制随机状态

    在一些使用的情况中，用户有可能想要将所有随机数生成器的“state”从一个给定的theano graph (例如， g1,和下面编译的函数f1一起的) 到另一个 graph (例如 g2,和函数f2一起的).。如果你想要从之前模型的pickled版本的参数中初始化模型的state，那么这个问题就会出现。 theano.tensor.shared\_randomstreams.RandomStreams 和 [theano.sandbox.rng\_mrg.MRG\_RandomStreams](http://deeplearning.net/software/theano/library/sandbox/rng_mrg.html#theano.sandbox.rng_mrg.MRG_RandomStreams) 可以通过复制state\_updates参数的元素来获得

    每次从一个RandomStreams对象中得到一个随机变量，一个元组就会被添入到 state\_updates 列表中。第一个元素就是一个共享变量，用来表示与这个具体的变量相关联的随机数生成器的状态，第二个元素用来表示对应于随机数生成过程(即RandomFunction{uniform}.0)的theano graph。

    一个关于如何“random states”可以从一个theano function迁移到另一个的例子如下：

1. **import theano**
2. **import numpy**
3. **import theano.tensor as T**
4. **from theano.sandbox.rng\_mrg import MRG\_RandomStreams**
5. **from theano.tensor.shared\_randomstreams import RandomStreams**
7. **class Graph():**
8. **def \_\_init\_\_(self, seed=123):**
9. self.rng = RandomStreams(seed)
10. self.y = self.rng.uniform(size=(1,))
12. g1 = Graph(seed=123)
13. f1 = theano.function([], g1.y)
15. g2 = Graph(seed=987)
16. f2 = theano.function([], g2.y)
18. **print 'By default, the two functions are out of sync.'**
19. **print 'f1() returns ', f1()**
20. **print 'f2() returns ', f2()**
22. **def copy\_random\_state(g1, g2):**
23. **if isinstance(g1.rng, MRG\_RandomStreams):**
24. g2.rng.rstate = g1.rng.rstate
25. **for (su1, su2) in zip(g1.rng.state\_updates, g2.rng.state\_updates):**
26. su2[0].set\_value(su1[0].get\_value())
28. **print 'We now copy the state of the theano random number generators.'**
29. copy\_random\_state(g1, g2)
30. **print 'f1() returns ', f1()**
31. **print 'f2() returns ', f2()**

    会有如下的输出：

1. # By default, the two functions are out of sync.
2. f1() returns  [ 0.72803009]
3. f2() returns  [ 0.55056769]
4. # We now copy the state of the theano random number generators.
5. f1() returns  [ 0.59044123]
6. f2() returns  [ 0.59044123]

### 5.5 其他随机分布

      见 [other distributions implemented](http://deeplearning.net/software/theano/library/tensor/raw_random.html#libdoc-tensor-raw-random).

### 5.6 其他实现

     这里有2个基于 [CURAND](http://deeplearning.net/software/theano/library/sandbox/cuda/op.html#module-theano.sandbox.cuda.rng_curand) 和 [MRG31k3p](http://deeplearning.net/software/theano/library/sandbox/rng_mrg.html#libdoc-rng-mrg).的两个实现。该RandomStream 只工作在CPU上，而 MRG31k3p 既能在CPU 上也能在GPU上。CURAND 只工作在 GPU上。

## 5.7 一个真实的例子：逻辑回归

      前面的内容会在下面这个更实际的例子中见到，而且该例子会被多次使用。

1. **import numpy**
2. **import theano**
3. **import theano.tensor as T**
4. rng = numpy.random
6. N = 400
7. feats = 784
8. D = (rng.randn(N, feats), rng.randint(size=N, low=0, high=2))
9. training\_steps = 10000
11. # Declare Theano symbolic variables声明theano符号变量
12. x = T.matrix("x")
13. y = T.vector("y")
14. w = theano.shared(rng.randn(feats), name="w")
15. b = theano.shared(0., name="b")
16. **print "Initial model:"**
17. **print w.get\_value(), b.get\_value()**
19. # Construct Theano expression graph构建theano表达式的图
20. p\_1 = 1 / (1 + T.exp(-T.dot(x, w) - b))   # Probability that target = 1
21. prediction = p\_1 > 0.5                    # The prediction thresholded
22. xent = -y \* T.log(p\_1) - (1-y) \* T.log(1-p\_1) # Cross-entropy loss function
23. cost = xent.mean() + 0.01 \* (w \*\* 2).sum()# The cost to minimize
24. gw, gb = T.grad(cost, [w, b])             # Compute the gradient of the cost
25. # (we shall return to this in a
26. # following section of this tutorial)
28. # Compile编译
29. train = theano.function(
30. inputs=[x,y],
31. outputs=[prediction, xent],
32. updates=((w, w - 0.1 \* gw), (b, b - 0.1 \* gb)))
33. predict = theano.function(inputs=[x], outputs=prediction)
35. # Train训练
36. **for i in range(training\_steps):**
37. pred, err = train(D[0], D[1])
39. **print "Final model:"**
40. **print w.get\_value(), b.get\_value()**
41. **print "target values for D:", D[1]**
42. **print "prediction on D:", predict(D[0])**

参考资料：

[1]官网：http://deeplearning.net/software/theano/tutorial/examples.html

## [Theano2.1.4-基础知识之图结构](http://www.cnblogs.com/shouhuxianjian/p/4590233.html)

来自：http://deeplearning.net/software/theano/tutorial/symbolic\_graphs.html

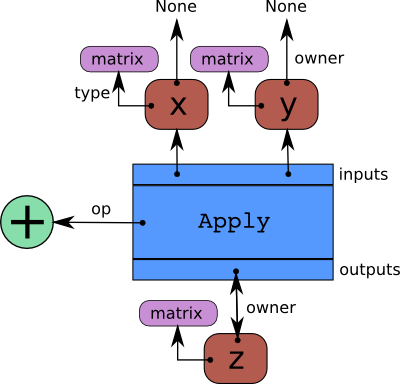
Graph Structures

    Theano是将符号数学化的计算表示成graphs。这些graphs是由将[Apply](http://deeplearning.net/software/theano/extending/graphstructures.html#apply) 和 [*Variable*](http://deeplearning.net/software/theano/extending/graphstructures.html#variable)节点内连接而组成的，它们是分别与函数的应用和数据相连接的。 操作是由 [Op](http://deeplearning.net/software/theano/extending/graphstructures.html#op) 实例所表示的，而数据类型是由 [Type](http://deeplearning.net/software/theano/extending/graphstructures.html#type) 实例所表示的。下面有一段代码和一个图表，该图表用来说明由这些代码所构建的结构。这些应该有助于你理解如何将这些片拟合到一起：

**代码：**

1. x = dmatrix('x')
2. y = dmatrix('y')
3. z = x + y

**图**



    箭头用来表示对指向的python对象的引用。这里的蓝色盒子是一个 [Apply](http://deeplearning.net/software/theano/extending/graphstructures.html#apply) 节点。红色盒子是 [Variable](http://deeplearning.net/software/theano/extending/graphstructures.html#variable) 节点。绿色圆圈是[Ops](http://deeplearning.net/software/theano/extending/graphstructures.html#op).。紫色盒子是 [*Types*](http://deeplearning.net/software/theano/extending/graphstructures.html#type)。

    在创建 [Variables](http://deeplearning.net/software/theano/extending/graphstructures.html#variable) 之后，对它们应用 [Apply](http://deeplearning.net/software/theano/extending/graphstructures.html#apply) [Ops](http://deeplearning.net/software/theano/extending/graphstructures.html#op) 从而得到更多的变量，并得到一个二分、有向、无环图。变量指向 Apply 节点的过程是用来表示函数通过它们的owner 域来生成它们 。这些Apply节点是通过它们的inputs和outputs域来得到它们的输入和输出变量的 (Apply 的实例同样包含指向他们outputs的引用的列表，不过这些指针不计算在这个graph内)。

   x 和 y 的owner 域的指向都是None是因为它们不是另一个计算的结果。如果它们中的一个是另一个计算的结果，那么owner域将会指向另一个的蓝色盒子比如 z 。

    注意到 Apply 实例的输出指向z ，而z.owner也指回Apply实例的。

# 一、一个显式的例子

    在这个例子中，我们将会对比两种定义相同graph的方式。首先，一段短代码用来构建一个表达式 (graph)，该正常的方式是大多数graph构建的时候自动使用的。第二种方式，我们将会使用一段较长的代码来干相同的事情，这里面没有用到任何的快捷方式，所以这可以让graph的构建变得很 explicit。

**1.1 Short example**

这就是通常的方式：

1. # create 3 Variables with owner = None
2. x = T.matrix('x')
3. y = T.matrix('y')
4. z = T.matrix('z')
6. # create 2 Variables (one for 'e', one intermediate for y\*z)
7. # create 2 Apply instances (one for '+', one for '\*')
8. e = x + y \* z

**1.2 Long example**

这就是显式的构建graph的方式：

1. **from** theano.tensor **import** add, mul, Apply, Variable, TensorType
3. # Instantiate a type that represents a matrix of doubles
4. float64\_matrix = TensorType(dtype = 'float64',              # double
5. broadcastable = (False, False)) # matrix
7. # We make the Variable instances we need.
8. x = Variable(type = float64\_matrix, name = 'x')
9. y = Variable(type = float64\_matrix, name = 'y')
10. z = Variable(type = float64\_matrix, name = 'z')
12. # This is the Variable that we want to symbolically represents y\*z
13. mul\_variable = Variable(type = float64\_matrix)
14. **assert** mul\_variable.owner **is** None
16. # Instantiate a symbolic multiplication
17. node\_mul = Apply(op = mul,
18. inputs = [y, z],
19. outputs = [mul\_variable])
20. # Fields 'owner' and 'index' are set by Apply
21. **assert** mul\_variable.owner **is** node\_mul
22. # 'index' is the position of mul\_variable in mode\_mul's outputs
23. **assert** mul\_variable.index == 0
25. # This is the Variable that we want to symbolically represents x+(y\*z)
26. add\_variable = Variable(type = float64\_matrix)
27. **assert** add\_variable.owner **is** None
29. # Instantiate a symbolic addition
30. node\_add = Apply(op = add,
31. inputs = [x, mul\_variable],
32. outputs = [add\_variable])
33. # Fields 'owner' and 'index' are set by Apply
34. **assert** add\_variable.owner **is** node\_add
35. **assert** add\_variable.index == 0
37. e = add\_variable
39. # We have access to x, y and z through pointers
40. **assert** e.owner.inputs[0] **is** x
41. **assert** e.owner.inputs[1] **is** mul\_variable
42. **assert** e.owner.inputs[1].owner.inputs[0] **is** y
43. **assert** e.owner.inputs[1].owner.inputs[1] **is** z

    注意到如何调用 Apply 来修改当 [*Variables*](http://deeplearning.net/software/theano/extending/graphstructures.html#variable)指向的输出是自身的时候的 owner 和 index 域，和如何改变在输出列表中占据的顺序位置。这整个机制建立的一个DAG  (有向无环图，Directed Acyclic Graph)，从而来表示计算的过程，graph可以在theano中被编译和优化。

# 二、自动包装

    在graph中所有的节点都必须是 Apply 或者 Result的实例，不过  <Op subclass>.make\_node() 通常会包装常量来满足这些约束。例如， **tensor.add()** 操作实例可以写成如下形式：

1. e = dscalar('x') + 1

构建如下的graph:

1. node = Apply(op = add,
2. inputs = [Variable(type = dscalar, name = 'x'),
3. Constant(type = lscalar, data = 1)],
4. outputs = [Variable(type = dscalar)])
5. e = node.outputs[0]

# 三、Graph Structures

    下面的部分概述了可以用在建立theano计算graph的结构的每个类型。这里介绍这几种结构： [Apply](http://deeplearning.net/software/theano/extending/graphstructures.html#apply), [Constant](http://deeplearning.net/software/theano/extending/graphstructures.html#constant), [Op](http://deeplearning.net/software/theano/extending/graphstructures.html#op), [Variable](http://deeplearning.net/software/theano/extending/graphstructures.html#variable) and [Type](http://deeplearning.net/software/theano/extending/graphstructures.html#type).

### 3.1 Apply

    一个Apply节点是一个内部节点的类型，用来表示theano中的 [*computation graph*](http://deeplearning.net/software/theano/glossary.html#term-graph) 。不同于 [Variable nodes](http://deeplearning.net/software/theano/extending/graphstructures.html#variable)，Apply 节点通常不是直接被最终用户操作的。。它们可以通过变量的onwer域来访问。

    Apply节点通常是Apply类的一个实例。它表示[Op](http://deeplearning.net/software/theano/extending/graphstructures.html#op) 在一个或多个输入上的应用，这里每个输入都是一个 [Variable](http://deeplearning.net/software/theano/extending/graphstructures.html#variable)。按照惯例，每个Op都有责任去了解如何从一个输入列表中构建一个Apply节点。因此， Apply节点可以从 Op和输入列表的基础上，通过调用 Op.make\_node(\*inputs)来得到。

    相比较于python语言来说， [Apply](http://deeplearning.net/software/theano/extending/graphstructures.html#apply) 节点是函数调用的theano版本，而 [Op](http://deeplearning.net/software/theano/extending/graphstructures.html#op) 是函数定义的theano版本。

    一个Apply实例有着下面三个很重要的域：:

**3.1.1 op**

[Op](http://deeplearning.net/software/theano/extending/graphstructures.html#op) 决定着在什么地方使用函数/转换。

**3.1.2 inputs**

[Variables](http://deeplearning.net/software/theano/extending/graphstructures.html#variable) 的列表，用来表示函数的参数。

**3.1.3 outputs**

[Variables](http://deeplearning.net/software/theano/extending/graphstructures.html#variable) 的列表，用来表示函数的返回值。

    Apply实例可以通过调用 gof.Apply(op, inputs, outputs)来创建。

#### 3.2 Op

      theano中的[Op](http://deeplearning.net/software/theano/extending/graphstructures.html#op) 是在某些类型的输入上定义一个具体的计算，并生成某些类型的输出。它等效于在大多数编程语言中的函数定义。从一个输入 [Variables](http://deeplearning.net/software/theano/extending/graphstructures.html#variable) 的列表和一个Op，你就可以建立一个 [Apply](http://deeplearning.net/software/theano/extending/graphstructures.html#apply) 节点，来表示Op 对输入的应用。

    理解一个Op(函数的定义)和Apply节点（函数的应用）之间的差别是很重要的。如果你使用theano的结构来解释python语言，写的代码就像 def f(x): ... ，将会对f 生成一个Op，然而如果像 a = f(x) 或者 g(f(4), 5) 那么就生成一个涉及到f Op的Apply节点。

#### 3.3 Type

    theano中的 [Type](http://deeplearning.net/software/theano/extending/graphstructures.html#type) 用来表示潜在数据对象上的一组约束。这些约束允许theano能够定制C 代码来处理它们，并对计算graph进行静态优化。例如， 在theano.tensor中的[irow](http://deeplearning.net/software/theano/library/tensor/basic.html#libdoc-tensor-creation) 类型会在数据上有以下约束：

1. 必须是 numpy.ndarray: isinstance(x, numpy.ndarray)的实例
2. 必须是32位整数的数组: str(x.dtype) == 'int32'
3. shape必须是1xN: len(x.shape) == 2 and x.shape[0] == 1

    在知道这些约束条件下，theano可以生成额外的C代码，声明正确的数据类型和基于维度上进行准确次数的循环。

     Theano的 [Type](http://deeplearning.net/software/theano/extending/graphstructures.html#type) 不等同于 Python 的type 或者 class。在Theano中， [irow](http://deeplearning.net/software/theano/library/tensor/basic.html#libdoc-tensor-creation) 和 [dmatrix](http://deeplearning.net/software/theano/library/tensor/basic.html#libdoc-tensor-creation) 都是使用 numpy.ndarray 来作为潜在的类型来进行计算和存储数据，然而他们都是不同的theano type。使用dmatrix时候的约束如下：

1. 必须是 numpy.ndarray: isinstance(x, numpy.ndarray)的实例
2. 必须是64位的浮点数数组: str(x.dtype) == 'float64'
3. shape必须是 MxN，在 M 或 N上都没有限制: len(x.shape) == 2

    这些约束不同于上面列出的irow的约束。

   在这些情况中，type可以完全对应于python的type，例如我们这里定义的double 类型，对应着python的float。不过这些不是必须的，除非指定了，所以当我们说“type”的时候，说的就是theano的type。

#### 3.4 Variable

[*Variable*](http://deeplearning.net/software/theano/extending/graphstructures.html#variable) 是当你使用theano的时候主要用到的数据结构。你操作的符号输入都是变量，而且通过各种Ops在输入上的使用得到的也是变量。例如，当输入下面的命令：

1. >>> x = theano.tensor.ivector()
2. >>> y = -x

    x 和 y 都是变量，即变量类的实例。x和y 的 [Type](http://deeplearning.net/software/theano/extending/graphstructures.html#type) 都是 theano.tensor.ivector.

     y 是计算的输出值对应的变量，而x是对应于输入的变量。计算的自身是通过另一个节点来表示的，一个[Apply](http://deeplearning.net/software/theano/extending/graphstructures.html#apply) 节点，和通过 y.owner来进行访问。

    更具体的说，theano中一个变量是是一个基本结构，用来表示在计算中某个具体的点上的基准。通常是类 **Variable** 或者是它的一个子类的实例。

变量 r 包含着4个重要的域：

**3.4.1 type**

[Type](http://deeplearning.net/software/theano/extending/graphstructures.html#type) 定义这可以在计算中使用的变量。

**3.4.2 owner**

t是为None 或者一个变量的 [Apply](http://deeplearning.net/software/theano/extending/graphstructures.html#apply) 节点的一个输出。

**3.4.3 index**

整数，例如 owner.outputs[index] 是 r (当 owner 为 None的时候忽略)

**3.4.4 name**

一个字符串，用于友好的打印和调试。

变量还有一个特殊的子类: [Constant](http://deeplearning.net/software/theano/extending/graphstructures.html#constant).

##### 3.4.5 Constant

    一个常量就是有着一个额外域data的 [Variable](http://deeplearning.net/software/theano/extending/graphstructures.html#variable) ，它只能被设置一次。当在计算graph中用作 [Op](http://deeplearning.net/software/theano/extending/graphstructures.html#op)[application](http://deeplearning.net/software/theano/extending/graphstructures.html#apply)的输入的时候，需要假设该输入总是常量的数据域部分。更进一步的说，需要假设 [Op](http://deeplearning.net/software/theano/extending/graphstructures.html#op) 不会修改该输入，也就是说一个常量是可以参与大量的优化的，比如在C中的常量内联，常量折叠等。

     在一个 [**function**](http://deeplearning.net/software/theano/library/compile/function.html#function.function)的输入列表中，常量是无需指定的。事实上，做的话还会引起一个异常。

## 四、Graph 结构的扩展

    当我们开始对一个theano函数进行编写的时候，我们需要计算一些额外的信息。这部分描述的是所提供的信息的一部分，不是所有都说清楚了，所以如果当你找不到的时候，可以 email theano的开发团队。

   该 graph是在编译的开始进行复制。所以在编译的时候修改不会影响到用户的graph。

    每个变量都接受的一个新域叫做clients。它是一个对图中使用过的变量的每个位置的引用列表， 。如果length为0，就意味着该变量没有被使用。每个被使用的位置都是通过包含2个元素的元组描述的。下面就是该对的两个类型：

* 第一个元素是一个 Apply节点.
* 第一个元素是字符串“output”，也就是说该函数的输出是这个变量。

在对的这两个类型中，元组的第二个元素是一个索引，例如: var.clients[\*][0].inputs[index] orfgraph.outputs[index] 就是这个变量。

1. **import** theano
2. v = theano.tensor.vector()
3. f = theano.function([v], (v+1).sum())
4. theano.printing.debugprint(f)
5. # Sorted list of all nodes in the compiled graph.
6. topo = f.maker.fgraph.toposort()
7. topo[0].outputs[0].clients
8. # [(Sum(Elemwise{add,no\_inplace}.0), 0)]
9. topo[1].outputs[0].clients
10. # [('output', 0)]
12. # An internal variable
13. var = topo[0].outputs[0]
14. client = var.clients[0]
15. client
16. # (Sum(Elemwise{add,no\_inplace}.0), 0)
17. type(client[0])
18. # <class 'theano.gof.graph.Apply'>
19. **assert** client[0].inputs[client[1]] **is** var
21. # An output of the graph
22. var = topo[1].outputs[0]
23. client = var.clients[0]
24. client
25. # ('output', 0)
26. **assert** f.maker.fgraph.outputs[client[1]] **is** var

参考资料：

[1] 官网：http://deeplearning.net/software/theano/extending/graphstructures.html#type

[**Theano2.1.5-基础知识之打印出theano的图**](http://www.cnblogs.com/shouhuxianjian/p/4590231.html)

来自：http://deeplearning.net/software/theano/tutorial/printing\_drawing.html

Printing/Drawing Theano graphs

    Theano提供的函数**theano.printing.pprint()** 和 [**theano.printing.debugprint()**](http://deeplearning.net/software/theano/library/printing.html#theano.printing.debugprint) 可以用来在编译前和后打印一个graph到终端上。 **pprint()** 该函数更紧凑而且更偏向于数学形式， **debugprint()** 更为的详细。 Theano同样提供**pydotprint()** 来生成一张有关该函数的图片。更详细的可以看看 [*printing – Graph Printing and Symbolic Print Statement*](http://deeplearning.net/software/theano/library/printing.html#libdoc-printing).

note：当打印theano函数的时候，有时候会比较难读懂。为了简化过程，可以禁止一些theano优化，只要使用theano的flag： optimizer\_excluding=fusion:inplace. 不要在工作执行的时候使用这个flag，这会使得graph更慢而且使用更多的内存。

    考虑逻辑回归的例子：

1. >>> **import** numpy
2. >>> **import** theano
3. >>> **import** theano.tensor as T
4. >>> rng = numpy.random
5. >>> # Training data
6. >>> N = 400
7. >>> feats = 784
8. >>> D = (rng.randn(N, feats).astype(theano.config.floatX), rng.randint(size=N,low=0, high=2).astype(theano.config.floatX))
9. >>> training\_steps = 10000
10. >>> # Declare Theano symbolic variables
11. >>> x = T.matrix("x")
12. >>> y = T.vector("y")
13. >>> w = theano.shared(rng.randn(feats).astype(theano.config.floatX), name="w")
14. >>> b = theano.shared(numpy.asarray(0., dtype=theano.config.floatX), name="b")
15. >>> x.tag.test\_value = D[0]
16. >>> y.tag.test\_value = D[1]
17. >>> # Construct Theano expression graph
18. >>> p\_1 = 1 / (1 + T.exp(-T.dot(x, w)-b)) # Probability of having a one
19. >>> prediction = p\_1 > 0.5 # The prediction that is done: 0 or 1
20. >>> # Compute gradients
21. >>> xent = -y\*T.log(p\_1) - (1-y)\*T.log(1-p\_1) # Cross-entropy
22. >>> cost = xent.mean() + 0.01\*(w\*\*2).sum() # The cost to optimize
23. >>> gw,gb = T.grad(cost, [w,b])
24. >>> # Training and prediction function
25. >>> train = theano.function(inputs=[x,y], outputs=[prediction, xent], updates=[[w, w-0.01\*gw], [b, b-0.01\*gb]], name = "train")
26. >>> predict = theano.function(inputs=[x], outputs=prediction, name = "predict")

**友好的打印结果：**

1. >>> theano.printing.pprint(prediction)
2. 'gt((TensorConstant{1} / (TensorConstant{1} + exp(((-(x \\dot w)) - b)))),
3. TensorConstant{0.5})'

**调试打印**

预编译图：

1. >>> theano.printing.debugprint(prediction)
2. Elemwise{gt,no\_inplace} [@A] ''
3. |Elemwise{true\_div,no\_inplace} [@B] ''
4. | |DimShuffle{x} [@C] ''
5. | | |TensorConstant{1} [@D]
6. | |Elemwise{add,no\_inplace} [@E] ''
7. |   |DimShuffle{x} [@F] ''
8. |   | |TensorConstant{1} [@D]
9. |   |Elemwise{exp,no\_inplace} [@G] ''
10. |     |Elemwise{sub,no\_inplace} [@H] ''
11. |       |Elemwise{neg,no\_inplace} [@I] ''
12. |       | |dot [@J] ''
13. |       |   |x [@K]
14. |       |   |w [@L]
15. |       |DimShuffle{x} [@M] ''
16. |         |b [@N]
17. |DimShuffle{x} [@O] ''
18. |TensorConstant{0.5} [@P]

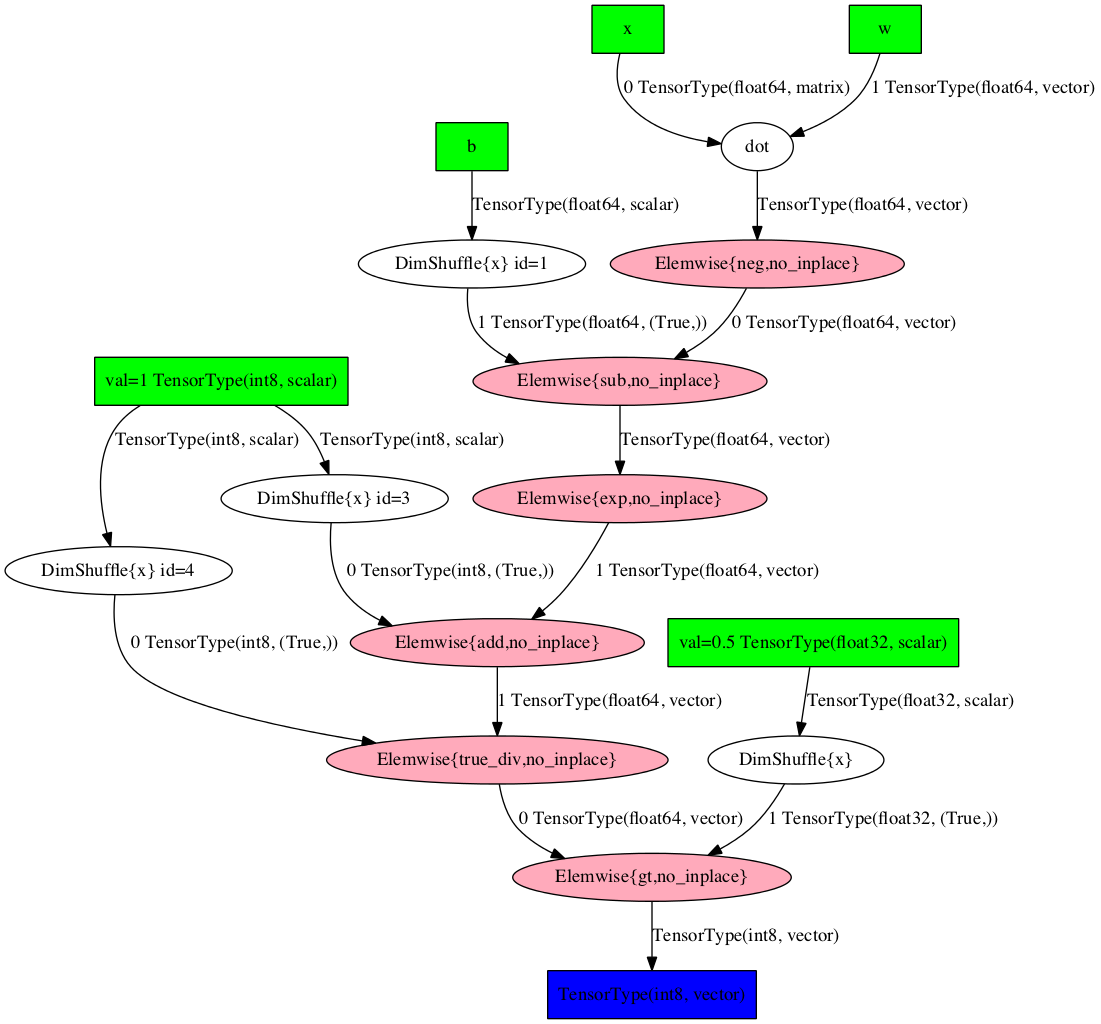
编译后的图:

1. >>> theano.printing.debugprint(predict)
2. Elemwise{Composite{GT(scalar\_sigmoid((-((-i0) - i1))), i2)}} [@A] ''   4
3. |CGemv{inplace} [@B] ''   3
4. | |Alloc [@C] ''   2
5. | | |TensorConstant{0.0} [@D]
6. | | |Shape\_i{0} [@E] ''   1
7. | |   |x [@F]
8. | |TensorConstant{1.0} [@G]
9. | |x [@F]
10. | |w [@H]
11. | |TensorConstant{0.0} [@D]
12. |InplaceDimShuffle{x} [@I] ''   0
13. | |b [@J]
14. |TensorConstant{(1,) of 0.5} [@K]

**graph的图片打印**

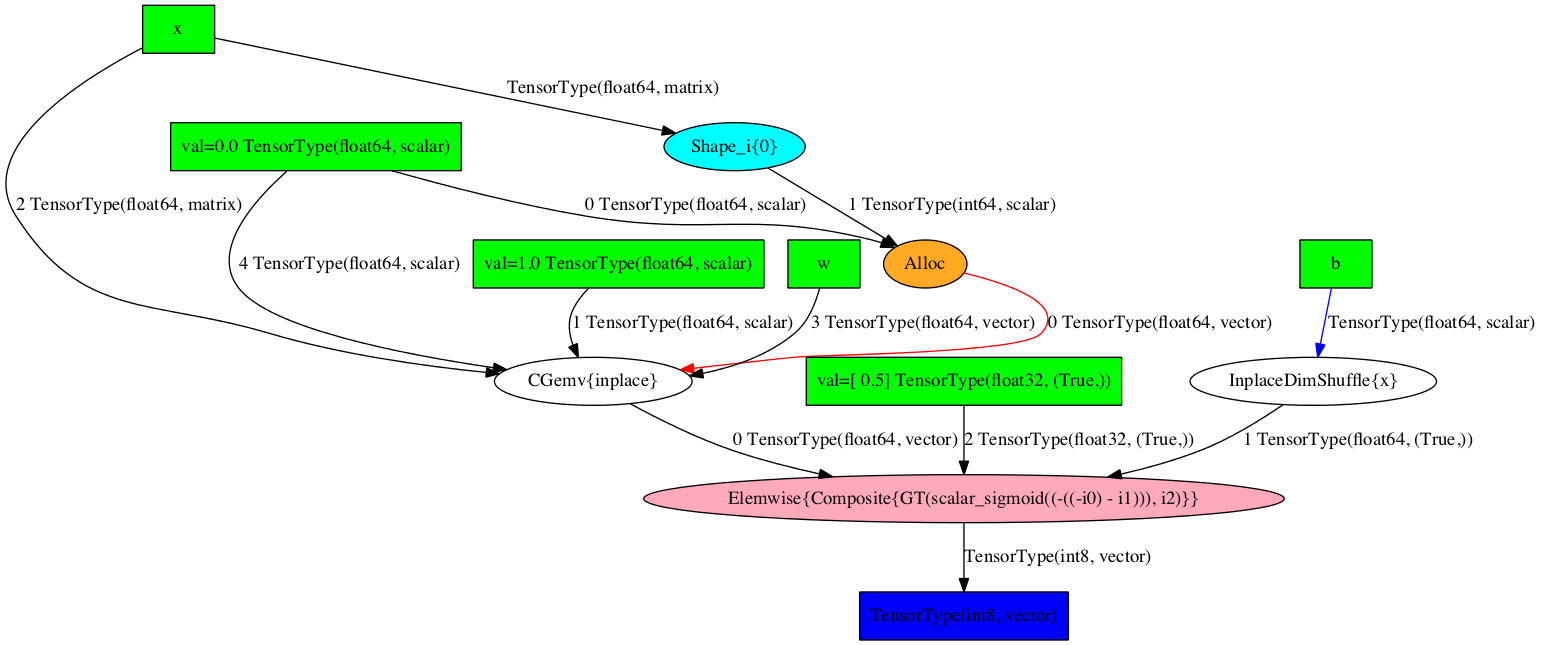
预编译图

1. >>> theano.printing.pydotprint(prediction, outfile="pics/logreg\_pydotprint\_prediction.png", var\_with\_name\_simple=True)
2. The output file **is** available at pics/logreg\_pydotprint\_prediction.png

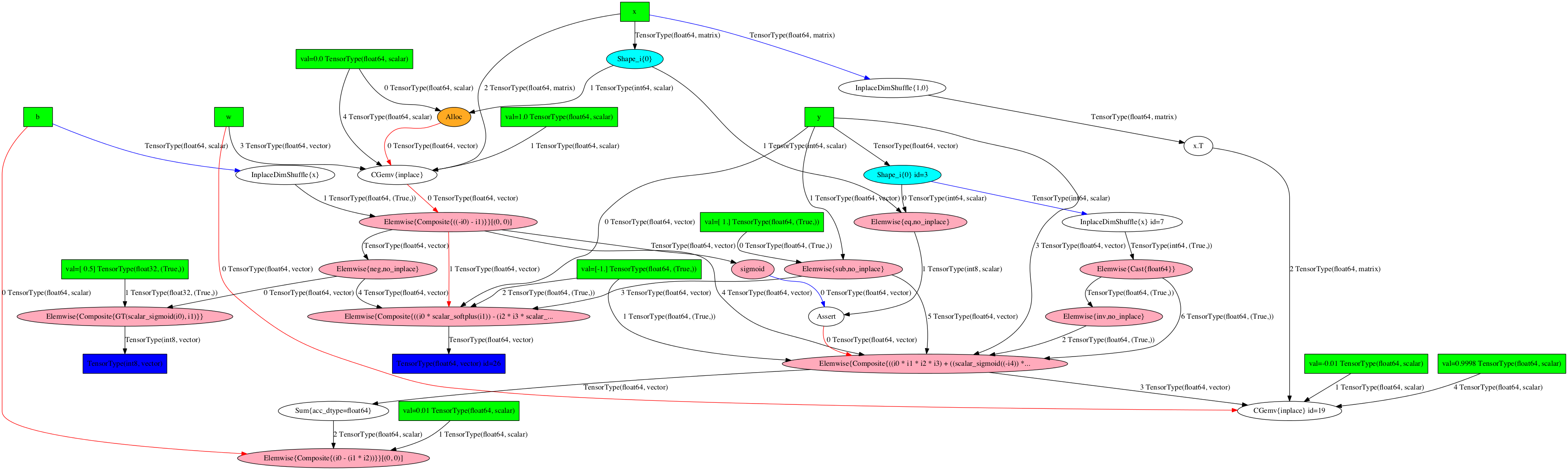


**编译后的图**

1. >>> theano.printing.pydotprint(predict, outfile="pics/logreg\_pydotprint\_predict.png", var\_with\_name\_simple=True)
2. The output file **is** available at pics/logreg\_pydotprint\_predict.png

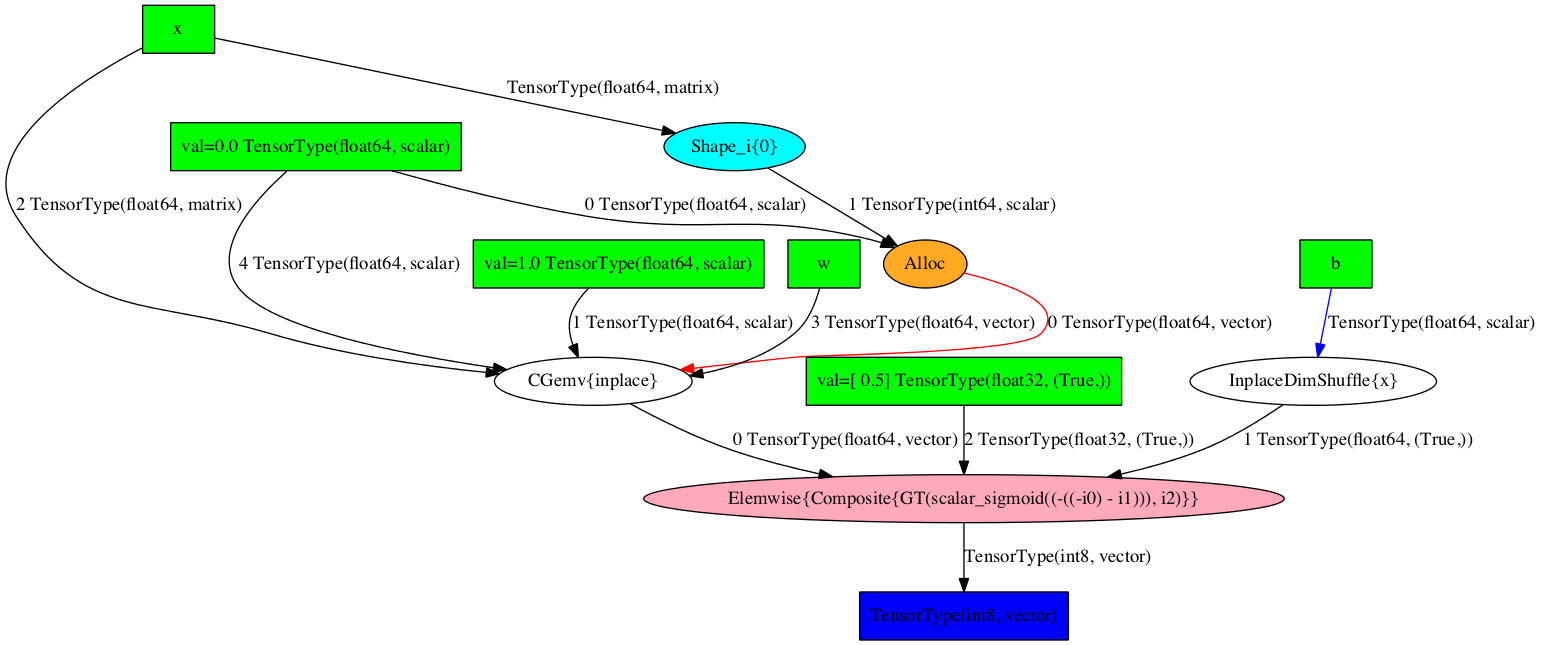
  
优化后的训练图：

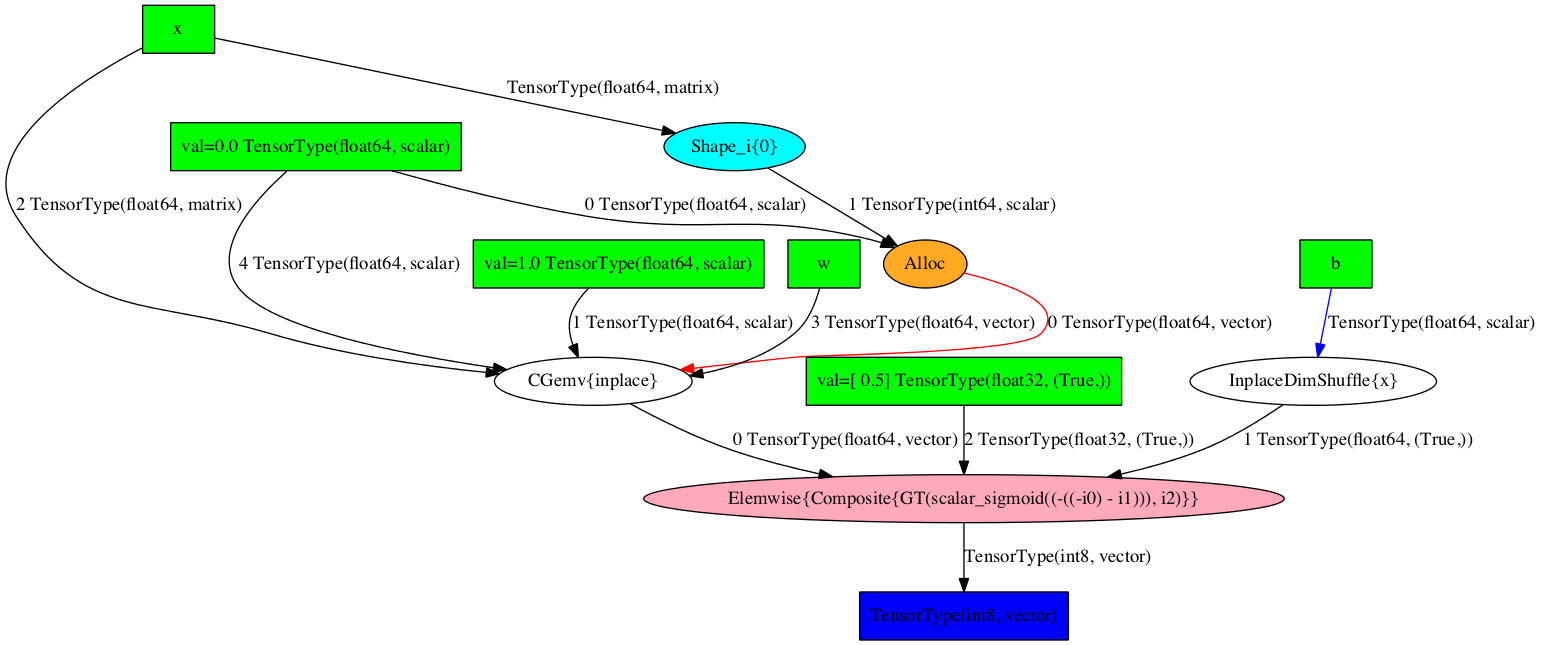
1. >>> theano.printing.pydotprint(train, outfile="pics/logreg\_pydotprint\_train.png", var\_with\_name\_simple=True)
2. The output file **is** available at pics/logreg\_pydotprint\_train.png

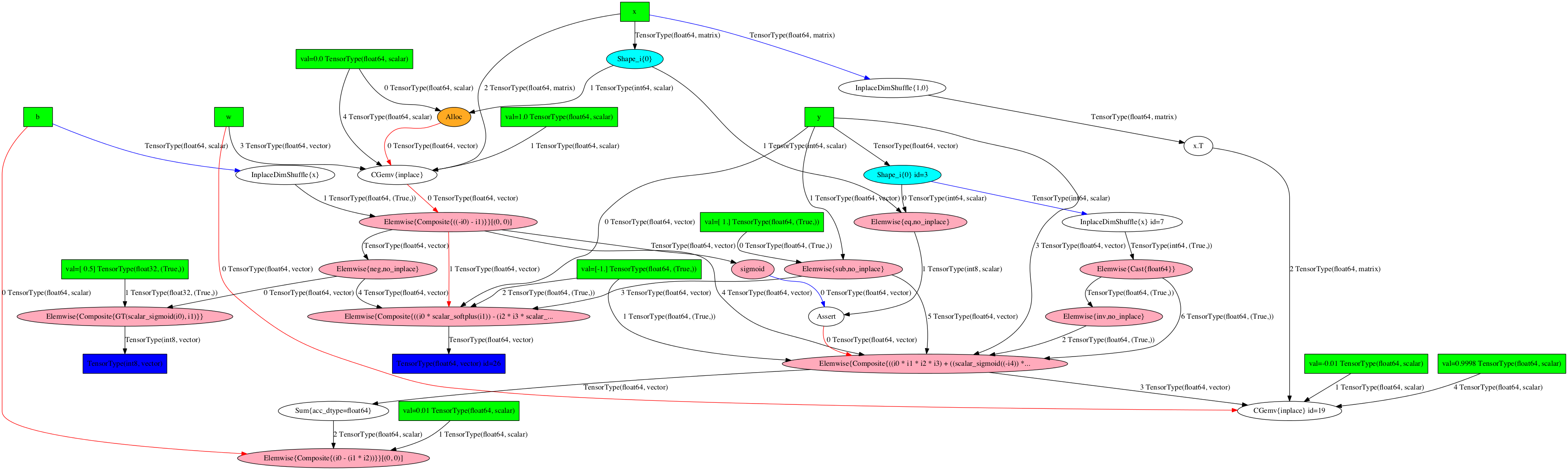
[](http://deeplearning.net/software/theano/_images/logreg_pydotprint_train2.png)

参考资料：

[1] 官网：http://deeplearning.net/software/theano/tutorial/printing\_drawing.html





[](http://deeplearning.net/software/theano/_images/logreg_pydotprint_train2.png)

## [Theano2.1.6-基础知识之在thenao中的求导](http://www.cnblogs.com/shouhuxianjian/p/4590230.html)

来自：http://deeplearning.net/software/theano/tutorial/gradients.html

Derivatives in Theano

# 一、计算梯度

    现在，让我们使用theano来做稍微更复杂的任务：创建一个函数，用来计算表达式y 关于它的参数x的导数。我们将会用到宏 T.grad 。例如，我们可以计算 x^2 关于 x的梯度。注意： d(x^2)/dx = 2 \cdot x.

    下面就是用来计算这个梯度的代码：

1. >>> **from** theano **import** pp
2. >>> x = T.dscalar('x')
3. >>> y = x \*\* 2
4. >>> gy = T.grad(y, x)
5. >>> pp(gy)  # print out the gradient prior to optimization
6. '((fill((x \*\* 2), 1.0) \* 2) \* (x \*\* (2 - 1)))'
7. >>> f = function([x], gy)
8. >>> f(4)
9. array(8.0)
10. >>> f(94.2)
11. array(188.40000000000001)

    在这个例子中，我们可以从pp(gy) 中看到我们在计算的符号梯度是正确的。 fill((x \*\* 2), 1.0) 意思是说创建一个和 x \*\* 2一样shape的矩阵，然后用1.0来填充。

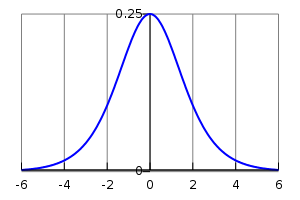
note：该优化器简化了符号梯度的表达式，你可以深挖编译后的函数的内部属性来了解细节。

pp(f.maker.fgraph.outputs[0])

'(2.0 \* x)'

在优化之后，在graph中只有一个 Apply节点，其输入是乘以2的。

    我们同样可以计算复杂表达式的梯度，例如由上面定义的逻辑函数。结果显示逻辑函数的梯度为： ds(x)/dx = s(x) \cdot (1 - s(x)).



该图是逻辑函数的梯度，x轴表示x的变化，y轴表示梯度 ds(x)/dx 。

1. >>> x = T.dmatrix('x')
2. >>> s = T.sum(1 / (1 + T.exp(-x)))
3. >>> gs = T.grad(s, x)
4. >>> dlogistic = function([x], gs)
5. >>> dlogistic([[0, 1], [-1, -2]])
6. array([[ 0.25      ,  0.19661193],
7. [ 0.19661193,  0.10499359]])

    通常来说，对于任何标量表达式， T.grad(s, w) 提供theano表达式来计算 \frac{\partial s}{\partial w}。这种方式下，甚至对于有着许多输入的函数来说，theano可以用来高效的计算符号微分 (正如 T.grad 返回的表达式可以在编译期间进行优化) ( [automatic differentiation](http://en.wikipedia.org/wiki/Automatic_differentiation) 有详细的描述关于符号微分的)。

note： T.grad 的第二个参数可以是一个列表，这种情况下，输出也同样是一个列表。在这两个列表中的顺序都是很重要的：输出列表的第 i 个元素是T.grad 的第一个参数关于第二个参数的列表的第 i 个元素的梯度。 T.grad 第一个参数必须是一个标量(其tensor size 为1）。更多有关T.grad的参数的语义的信息和实现的细节，可以参考库的 [this](http://deeplearning.net/software/theano/library/gradient.html#libdoc-gradient) 部分。

在内部微分的工作的信息可以在更高级的教程 [Extending Theano](http://deeplearning.net/software/theano/extending/index.html#extending)中找到。

# 二、计算Jacobian

    在theano中，术语 Jacobian 指定为张量包含函数的输出关于输入的第一个偏导数。 （在数学中这就是所谓的Jacobian矩阵） Theano 实现宏[**theano.gradient.jacobian()**](http://deeplearning.net/software/theano/library/gradient.html#theano.gradient.jacobian) 所需要的就是计算Jacobian。下面部分就是解释如何手动去完成它：

    为了手动计算一些函数 y 关于一些参数 x 的Jacobian，我们需要使用 scan。即在y 中使用循环来遍历所有元素，然后计算 y[i] 关于x 的梯度。

note：scan 是theano中一个通用的操作，可以以符号方式写出各种递归等式。然而生成一个符号循环是很难的（而且还需要为了性能而去优化它们） ，所以需要努力提升scan.的效果。在后面会接着说 [scan](http://deeplearning.net/software/theano/tutorial/loop.html#tutloop) 的。

1. >>> x = T.dvector('x')
2. >>> y = x \*\* 2
3. >>> J, updates = theano.scan(**lambda** i, y,x : T.grad(y[i], x), sequences=T.arange(y.shape[0]), non\_sequences=[y,x])
4. >>> f = function([x], J, updates=updates)
5. >>> f([4, 4])
6. array([[ 8.,  0.],
7. [ 0.,  8.]])

    在该代码中所做的就是生成一个int类型的序列，通过使用T.arange来使得其中从0到 y.shape[0] 。然后我们对这个序列进行循环，然后在每一步，灭我们计算元素 y[i]关于x 的梯度。scan 可以自动的连接所有的这些列，生成一个对应于jacobian的矩阵。

note：在使用T.grad的时候记得也有一些陷阱的。 其中一个就是你没法和这样theano.scan(lambda y\_i,x: T.grad(y\_i,x), sequences=y, non\_sequences=x)重写jacobin的上述表达式，，尽管从文档上看scan是可以的。原因在于 y\_i 不再试x的函数了，而 y[i]仍然是。

# 三、计算Hessian

    在theano中，术语Hessian 与数学上的概念没差：是一个矩阵，其中包含着标量输出和向量输入的函数的二阶偏导数。Theano 实现宏[**theano.gradient.hessian()**](http://deeplearning.net/software/theano/library/gradient.html#theano.gradient.hessian) 所要做的就是计算Hessian。下面的部分就是介绍如何手动完成。

    你可以可jacobian一样相似的计算Hessian。唯一的差别在于，我们通过计算T.grad(cost,x)的jacobian来代替计算一些表达式y 的jacobian，所以计算的cost是标量的。

1. >>> x = T.dvector('x')
2. >>> y = x \*\* 2
3. >>> cost = y.sum()
4. >>> gy = T.grad(cost, x)
5. >>> H, updates = theano.scan(**lambda** i, gy,x : T.grad(gy[i], x), sequences=T.arange(gy.shape[0]), non\_sequences=[gy, x])
6. >>> f = function([x], H, updates=updates)
7. >>> f([4, 4])
8. array([[ 2.,  0.],
9. [ 0.,  2.]])

# 四、Jacobian乘以一个向量

    有时候我们需要将算法表示成jacobinas乘以向量，或者向量乘以jacobinans。相比较于评估jacobian，然后做乘法，可以直接计算合适的结果从而避免对jacobian的实际计算。这可以带来明显的性能的提升。一个这样的算法可以在下面的文献中找到：

* Barak A. Pearlmutter, “Fast Exact Multiplication by the Hessian”, Neural Computation, 1994

    然而在实际中，我们想要theano能够自动的识别这些模式，不过以通常的方式来实现这样的优化是非常难的。所以，我们提供了特别的函数来应对这些问题：

### R-operator

    R操作符是用来评估介于一个jacobian和一个向量之间的乘积的，即 \frac{\partial f(x)}{\partial x} v. 该式子可以扩展成当x是一个矩阵，或者一个张量的形式，这种情况下，jacobian就变成了一个张量，然后乘积就变成了某种张量的积。因为在实际中，我们最后是需要计算权重矩阵这样的表达式的，theano支持这种操作的更通用形式。为了评估表达式y的R 操作，（关于x的），使用v乘以jacobian，你需要做类似下面的事情：

1. >>> W = T.dmatrix('W')
2. >>> V = T.dmatrix('V')
3. >>> x = T.dvector('x')
4. >>> y = T.dot(x, W)
5. >>> JV = T.Rop(y, W, V)
6. >>> f = theano.function([W, V, x], JV)
7. >>> f([[1, 1], [1, 1]], [[2, 2], [2, 2]], [0,1])
8. array([ 2.,  2.])

实现Rop的操作列表[List](http://deeplearning.net/software/theano/library/tensor/basic.html#r-op-list) 。

### L-operator

    相似于R-操作，L-操作 会计算一个行向量乘积，其数学上的形式为 v \frac{\partialf(x)}{\partial x}。该L-操纵 同样支持通用的张量 (不只是向量)。相思的，它可以按照下面形式实现：

1. >>> W = T.dmatrix('W')
2. >>> v = T.dvector('v')
3. >>> x = T.dvector('x')
4. >>> y = T.dot(x, W)
5. >>> VJ = T.Lop(y, W, v)
6. >>> f = theano.function([v,x], VJ)
7. >>> f([2, 2], [0, 1])
8. array([[ 0.,  0.],
9. [ 2.,  2.]])

note：v, 在L操作和R操作中是不同的。对于L操作来说，该 v 需要有着和输出一样的shape，然而，R操作需要和输入参数一样的shape。更进一步说，这两个操作的结果是不同的。L操作的结果有着和输入参数一样的shape，而R操作有着和输出一样的shape。

# 五、Hessian乘以一个向量

    如果你需要计算Hessian 乘以一个向量，你就需要用到上面定义的操作，它们通常比实际计算准确的Hessian，然后计算乘积更高效。因为Hessian矩阵的对称性，你可以用两种方式得到相同的结果，虽然这些选择也许会有不同的性能。因此，我们建议在使用它们之前先, 先对它们进行分析：

1. >>> x = T.dvector('x')
2. >>> v = T.dvector('v')
3. >>> y = T.sum(x \*\* 2)
4. >>> gy = T.grad(y, x)
5. >>> vH = T.grad(T.sum(gy \* v), x)
6. >>> f = theano.function([x, v], vH)
7. >>> f([4, 4], [2, 2])
8. array([ 4.,  4.])

或者使用R操作：

1. >>> x = T.dvector('x')
2. >>> v = T.dvector('v')
3. >>> y = T.sum(x \*\* 2)
4. >>> gy = T.grad(y, x)
5. >>> Hv = T.Rop(gy, x, v)
6. >>> f = theano.function([x, v], Hv)
7. >>> f([4, 4], [2, 2])
8. array([ 4.,  4.])

备注：

* grad 函数是符号化的工作的：它接受和返回theano变量。
* grad 可以和宏相比较，因为它可以重复使用
* 标量损失只能被直接通过grad进行处理。数组可以通过重复应用的形式来解决
* 内建的函数可以高效的计算向量乘以jacobian和向量乘以Hessian
* 优化需要高效的计算全jacobian和Hessian矩阵，以及jacobian乘以向量。

参考资料：

[1]官网：http://deeplearning.net/software/theano/tutorial/gradients.html

## [Theano2.1.7-基础知识之设置的配置和编译模式](http://www.cnblogs.com/shouhuxianjian/p/4590229.html)

来自：http://deeplearning.net/software/theano/tutorial/modes.html

Configuration Settings and Compiling Modes

# 一、配置

    该 config 模块包含了好几个属性用来修改theano的行为。许多属性会在导入theano模块的时候被检查的，其中一些属性是被假定成只读形式的。约定俗成，在config模块中的属性不应该在用户的代码中被修改。

    Theano的代码对这些属性都有默认值的，不过你可以从你的 .theanorc 文件中对它们进行覆盖，然而 [**THEANO\_FLAGS**](http://deeplearning.net/software/theano/library/config.html#envvar-THEANO_FLAGS) 环境变量又会覆盖这些值。

优先级顺序如下：

1. 对theano.config.<property>的赋值。
2. 在[**THEANO\_FLAGS**](http://deeplearning.net/software/theano/library/config.html#envvar-THEANO_FLAGS)中的赋值
3. 在 .theanorc file (或者是在 [**THEANORC**](http://deeplearning.net/software/theano/library/config.html#envvar-THEANORC)中指定的文件)文件中的赋值。

    你可以在任何时候通过theano.config打印出当前的配置。例如，为了查看所有激活的配置变量的列表，输入下面的命令：

1. python -c 'import theano; print theano.config' | less

更详细的，请看库中的 [Configuration](http://deeplearning.net/software/theano/library/config.html#libdoc-config) 。

# 二、练习

考虑逻辑回归：

1. **import** numpy
2. **import** theano
3. **import** theano.tensor as T
4. rng = numpy.random
6. N = 400
7. feats = 784
8. D = (rng.randn(N, feats).astype(theano.config.floatX),
9. rng.randint(size=N,low=0, high=2).astype(theano.config.floatX))
10. training\_steps = 10000
12. # Declare Theano symbolic variables
13. x = T.matrix("x")
14. y = T.vector("y")
15. w = theano.shared(rng.randn(feats).astype(theano.config.floatX), name="w")
16. b = theano.shared(numpy.asarray(0., dtype=theano.config.floatX), name="b")
17. x.tag.test\_value = D[0]
18. y.tag.test\_value = D[1]
19. #print "Initial model:"
20. #print w.get\_value(), b.get\_value()
22. # Construct Theano expression graph
23. p\_1 = 1 / (1 + T.exp(-T.dot(x, w)-b)) # Probability of having a one
24. prediction = p\_1 > 0.5 # The prediction that is done: 0 or 1
25. xent = -y\*T.log(p\_1) - (1-y)\*T.log(1-p\_1) # Cross-entropy
26. cost = xent.mean() + 0.01\*(w\*\*2).sum() # The cost to optimize
27. gw,gb = T.grad(cost, [w,b])
29. # Compile expressions to functions
30. train = theano.function(
31. inputs=[x,y],
32. outputs=[prediction, xent],
33. updates={w:w-0.01\*gw, b:b-0.01\*gb},
34. name = "train")
35. predict = theano.function(inputs=[x], outputs=prediction,
36. name = "predict")
38. **if** any([x.op.\_\_class\_\_.\_\_name\_\_ **in** ['Gemv', 'CGemv', 'Gemm', 'CGemm'] **for** x **in**
39. train.maker.fgraph.toposort()]):
40. **print** 'Used the cpu'
41. **elif** any([x.op.\_\_class\_\_.\_\_name\_\_ **in** ['GpuGemm', 'GpuGemv'] **for** x **in**
42. train.maker.fgraph.toposort()]):
43. **print** 'Used the gpu'
44. **else**:
45. **print** 'ERROR, not able to tell if theano used the cpu or the gpu'
46. **print** train.maker.fgraph.toposort()
48. **for** i **in** range(training\_steps):
49. pred, err = train(D[0], D[1])
50. #print "Final model:"
51. #print w.get\_value(), b.get\_value()
53. **print** "target values for D"
54. **print** D[1]
56. **print** "prediction on D"
57. **print** predict(D[0])

    修改这个例子然后在cpu（默认情况）上执行，使用floatX=float32，然后使用计时命令 time python file.py（该命令在win8下无法执行）。保存你的代码，稍后还会用到。

note：

* 在代码中使用theano的flag floatX=float32 (通过theano.config.floatX来配置) 。
* 在存储到共享变量之前先Cast输入到一个共享变量中
* 避免本来将int32 cast成float32的，自动cast成float64.
  + 在代码中手动插入cast 或者使用[u]int{8,16}.
  + 在均值操作上手动插入cast (这会涉及到除以length，其中length是一个int64的类型).
  + 注意到一个新的casting机制现在在开发。

答案（[**Solution**](http://deeplearning.net/software/theano/_downloads/modes_solution_1.py)）：

1. #!/usr/bin/env python
2. # Theano tutorial
3. # Solution to Exercise in section 'Configuration Settings and Compiling Modes'
5. **from** \_\_future\_\_ **import** print\_function
6. **import** numpy
7. **import** theano
8. **import** theano.tensor as tt
10. theano.config.floatX = 'float32'
12. rng = numpy.random
14. N = 400
15. feats = 784
16. D = (rng.randn(N, feats).astype(theano.config.floatX),
17. rng.randint(size=N, low=0, high=2).astype(theano.config.floatX))
18. training\_steps = 10000
20. # Declare Theano symbolic variables
21. x = tt.matrix("x")
22. y = tt.vector("y")
23. w = theano.shared(rng.randn(feats).astype(theano.config.floatX), name="w")
24. b = theano.shared(numpy.asarray(0., dtype=theano.config.floatX), name="b")
25. x.tag.test\_value = D[0]
26. y.tag.test\_value = D[1]
27. #print "Initial model:"
28. #print w.get\_value(), b.get\_value()
30. # Construct Theano expression graph
31. p\_1 = 1 / (1 + tt.exp(-tt.dot(x, w) - b))  # Probability of having a one
32. prediction = p\_1 > 0.5  # The prediction that is done: 0 or 1
33. xent = -y \* tt.log(p\_1) - (1 - y) \* tt.log(1 - p\_1)  # Cross-entropy
34. cost = tt.cast(xent.mean(), 'float32') + \
35. 0.01 \* (w \*\* 2).sum()  # The cost to optimize
36. gw, gb = tt.grad(cost, [w, b])
38. # Compile expressions to functions
39. train = theano.function(
40. inputs=[x, y],
41. outputs=[prediction, xent],
42. updates={w: w - 0.01 \* gw, b: b - 0.01 \* gb},
43. name="train")
44. predict = theano.function(inputs=[x], outputs=prediction,
45. name="predict")
47. **if** any([x.op.\_\_class\_\_.\_\_name\_\_ **in** ['Gemv', 'CGemv', 'Gemm', 'CGemm'] **for** x **in**
48. train.maker.fgraph.toposort()]):
49. **print**('Used the cpu')
50. **elif** any([x.op.\_\_class\_\_.\_\_name\_\_ **in** ['GpuGemm', 'GpuGemv'] **for** x **in**
51. train.maker.fgraph.toposort()]):
52. **print**('Used the gpu')
53. **else**:
54. **print**('ERROR, not able to tell if theano used the cpu or the gpu')
55. **print**(train.maker.fgraph.toposort())
57. **for** i **in** range(training\_steps):
58. pred, err = train(D[0], D[1])
59. #print "Final model:"
60. #print w.get\_value(), b.get\_value()
62. **print**("target values for D")
63. **print**(D[1])
65. **print**("prediction on D")
66. **print**(predict(D[0]))

# 三、模式

    每次 [**theano.function**](http://deeplearning.net/software/theano/library/compile/function.html#function.function) 被调用的时候，介于输入和输出之间的theano变量之间的符号关系是会被优化和编译的。编译的方式是由mode参数所控制的。

Theano通过名字定义的以下模型：

* 'FAST\_COMPILE': 只使用一点graph优化，只用python实现。
* 'FAST\_RUN': 使用所有的优化，并在可能的地方使用c实现。
* 'DebugMode: 验证所有优化的正确性，对比c和pytho的实现。该模式可能会花比其他模式下更长的时间，不过却能验证几种不同的问题。
* 'ProfileMode' (弃用): 和FAST\_RUN一样的优化，不过打印出一些分析信息
* 默认模式是 FAST\_RUN,，不过可以通过配置变量 [**config.mode**](http://deeplearning.net/software/theano/library/config.html#config.mode)来改变，这可以通过传递关键参数给[**theano.function**](http://deeplearning.net/software/theano/library/compile/function.html#function.function)来重写该值。

| **short name** | **Full constructor** | **What does it do?** |
| --- | --- | --- |
| FAST\_COMPILE | compile.mode.Mode(linker='py',optimizer='fast\_compile') | 只用Python实现,快速和简单的graph转换 |
| FAST\_RUN | compile.mode.Mode(linker='cvm',optimizer='fast\_run') | 在可以的地方用C实现，使用所有的graph转换技术 |
| DebugMode | compile.debugmode.DebugMode() | 两种实现方式，使用所有的graph转换技术 |
| ProfileMode | compile.profilemode.ProfileMode() | 弃用，在可以的地方c实现，所有的graph转换技术，打印profile 信息 |

note：对于调试的目的来说，还有一个 MonitorMode 。它可以用来以step的方式来查看函数的执行，更详细的看 [the debugging FAQ](http://deeplearning.net/software/theano/tutorial/debug_faq.html#faq-monitormode)

# 四、连接器

   模式是有2个部分组成的：一个优化器和一个连接器。许多模式，例如 ProfileMode 和 DebugMode, 在优化器和连接器上增加逻辑。 ProfileMode 和DebugMode使用它们自己的连接器。

    可以通过theano flag [**config.linker**](http://deeplearning.net/software/theano/library/config.html#config.linker)来选择使用哪个连接器。这里是一个不同连接器的对比表：

| **linker** | **gc**[**[1]**](http://deeplearning.net/software/theano/tutorial/modes.html#gc) | **Raise error by op** | **Overhead** | **Definition** |
| --- | --- | --- | --- | --- |
| cvm | yes | yes | “++” | 和 c | py一样， but the runtime algo to execute the code is in c |
| cvm\_nogc | no | yes | “+” | 和 cvm一样，不过没有gc |
| c|py [[2]](http://deeplearning.net/software/theano/tutorial/modes.html#cpy1) | yes | yes | “+++” | 尝试使用 C code，如果没有有关op 的c代码，那就使用Python的 |
| c|py\_nogc | no | yes | “++” | 和 c|py一样，不过没有 gc |
| c | no | yes | “+” | 只用 C代码 (如果对op没有可用的c代码，抛出错误) |
| py | yes | yes | “+++” | 只用Python代码 |
| ProfileMode | no | no | “++++” | (弃用) 计算一些额外的profiling信息 |
| DebugMode | no | yes | VERY HIGH | 在theano的计算上进行许多检查 |

[1]  在计算的时候对中间的值采用垃圾回收。不然，为了不要重新分配内存，和更少的重写（意味着更快），被ops使用的内存空间将一直保存在theano的函数调用中。

[2] 默认。

更多详细信息，查看库中的 [Mode](http://deeplearning.net/software/theano/library/compile/mode.html#libdoc-compile-mode) 部分。

# 五、使用调试模式

   通常来说，你应该使用 FAST\_RUN 或者 FAST\_COMPILE 模式，首先在使用调试模式的时候（mode='DebugMode）运行你的代码的时候，这很有用 (特别是当你在定义新的表达式或新的优化的时候) 。调试模式是设计用来运行一些自我检查和断言，有助于诊断可能的编码错误导致的不正确输出。 。注意到DebugMode 比 FAST\_RUN 或 FAST\_COMPILE 要慢，所以只在开发的时候使用该模式 (不要当在一个集群上运行1000 进程的时候用).

调试模式按如下方式使用：

1. x = T.dvector('x')
3. f = theano.function([x], 10 \* x, mode='DebugMode')
5. f([5])
6. f([0])
7. f([7])

    如果检测到任何问题，DebugMode 将会抛出一个异常来指定出错的信息，不论是在调用的时候(f(5))还是编译的时候(f = theano.function(x, 10 \* x, mode='DebugMode'))。这些异常不应该被忽略，和你的当地的theano guru谈谈或者当异常没法搞定的时候记得给使用者发邮件

    许多种错误只能只有当某些输入值结合的时候才会被检测到。在上面的例子中，没有方法保证说一个函数的调用，例如f(-1)不会引起问题，DebugMode不是银弹（有本软件工程的书就叫做《没有银弹》）。

    如果你使用构造器（见DebugMode）来实例化 DebugMode 而不是使用关键字 DebugMode ，你就能通过构造器的参数来配置它的行为。而DebugMode的关键字版本是相当严格的 (通过使用 mode='DebugMode'来得到) 。

更详细的，见库的*[DebugMode](http://deeplearning.net/software/theano/library/compile/debugmode.html" \l "debugmode" \t "_blank)* 。

# 六、ProfileMode

note：ProfileMode 被弃用了，使用 [**config.profile**](http://deeplearning.net/software/theano/library/config.html#config.profile) 来代替的。

    在检查错误的同事，另一个重要的任务就是profile你的代码。对于thean使用的一个特殊的模式叫做ProfileMode，它是用来作为参数传递给 [**theano.function**](http://deeplearning.net/software/theano/library/compile/function.html#function.function)的。使用该模式是一个三步的过程。

note：为了切换到相应的默认情况下，设置theano 的flag [**config.mode**](http://deeplearning.net/software/theano/library/config.html#config.mode) 为ProfileMode。在这种情况下，当python的进程存在的时候，它会自动的打印profiling信息到标准输出端口上。

T每个apply节点的输出的内存profile可以被theano 的flag **config.ProfileMode.profile\_memory**所启用。

更详细的，看看库中 [ProfileMode](http://deeplearning.net/software/theano/library/compile/profilemode.html#profilemode) 的部分。

# 七、创建一个ProfileMode实例

首先，创建一个ProfileMode实例：

1. >>> **from** theano **import** ProfileMode
2. >>> profmode = theano.ProfileMode(optimizer='fast\_run', linker=theano.gof.OpWiseCLinker())

     ProfileMode的构造器将一个优化器和一个连接器作为输入。使用哪个优化器和连接器是由应用所决定的。例如，一个用户想要只profile python的实现，就应该使用gof.PerformLinker (或者 “py” for short)。在另一方面，一个用户想要使用c实现来profile他的graph，那么久应该使用 gof.OpWiseCLinker (or “c|py”)。为了测试你代码的速度，我们推荐使用fast\_run 优化器和 gof.OpWiseCLinker 连接器。

# 八、用ProfileMode来编译graph

    一旦ProfileMode实例创建好了，通过指定模式的参数来简化编译你的graph，就和平常一样：

1. >>> # with functions
2. >>> f = theano.function([input1,input2],[output1], mode=profmode)

# 九、检索时间信息

   一旦你的graph编译好了，简单的运行你希望profile的程序或操作，然后调用 profmode.print\_summary()。 这会给你提供合适的时间信息，用来指明你的graph的哪个地方最耗时。这最好通过一个例子来说明，我们接着使用逻辑回归的例子吧。

    使用 ProfileMode来编译模块，然后调用profmode.print\_summary() 来生成下面的输出：

1. """
2. ProfileMode.print\_summary()
3. ---------------------------
5. local\_time 0.0749197006226 (Time spent running thunks)
6. Apply-wise summary: <fraction of local\_time spent at this position> (<Apply position>, <Apply Op name>)
7. 0.069   15      \_dot22
8. 0.064   1       \_dot22
9. 0.053   0       InplaceDimShuffle{x,0}
10. 0.049   2       InplaceDimShuffle{1,0}
11. 0.049   10      mul
12. 0.049   6       Elemwise{ScalarSigmoid{output\_types\_preference=<theano.scalar.basic.transfer\_type object at 0x171e650>}}[(0, 0)]
13. 0.049   3       InplaceDimShuffle{x}
14. 0.049   4       InplaceDimShuffle{x,x}
15. 0.048   14      Sum{0}
16. 0.047   7       sub
17. 0.046   17      mul
18. 0.045   9       sqr
19. 0.045   8       Elemwise{sub}
20. 0.045   16      Sum
21. 0.044   18      mul
22. ... (remaining 6 Apply instances account **for** 0.25 of the runtime)
23. Op-wise summary: <fraction of local\_time spent on this kind of Op> <Op name>
24. 0.139   \* mul
25. 0.134   \* \_dot22
26. 0.092   \* sub
27. 0.085   \* Elemwise{Sub{output\_types\_preference=<theano.scalar.basic.transfer\_type object at 0x1779f10>}}[(0, 0)]
28. 0.053   \* InplaceDimShuffle{x,0}
29. 0.049   \* InplaceDimShuffle{1,0}
30. 0.049   \* Elemwise{ScalarSigmoid{output\_types\_preference=<theano.scalar.basic.transfer\_type object at 0x171e650>}}[(0, 0)]
31. 0.049   \* InplaceDimShuffle{x}
32. 0.049   \* InplaceDimShuffle{x,x}
33. 0.048   \* Sum{0}
34. 0.045   \* sqr
35. 0.045   \* Sum
36. 0.043   \* Sum{1}
37. 0.042   \* Elemwise{Mul{output\_types\_preference=<theano.scalar.basic.transfer\_type object at 0x17a0f50>}}[(0, 1)]
38. 0.041   \* Elemwise{Add{output\_types\_preference=<theano.scala

参考资料：

[1]官网：http://deeplearning.net/software/theano/tutorial/modes.html

## [Theano2.1.8-基础知识之装载和保存](http://www.cnblogs.com/shouhuxianjian/p/4590228.html)

来自：http://deeplearning.net/software/theano/tutorial/loading\_and\_saving.html

loading and saving

    Python的标准方式来保存类实例和装载的方式是 [pickle](http://docs.python.org/library/pickle.html) 机制。许多theano对象可以被pickle序列化（和反序列化）。然而，pickles的限制是它没法当类实例被序列化的时候，保存类的代码或者数据。所以，重新装载之前的类创建的对象变得非常有问题。

    所以，你想要使用另一个不同的机制，这取决于你在保存和重新装载之间间隔的时间的大小。对于短期来说(例如临时文件或者网络迁移)，将theano的对象或者类进行pickling是可能的。对于长期来说 (例如从一个试验中保存模型) ，就不要依赖于 pickled Theano对象了；我们推荐在其他python的程序过程中装载和保存这个底层的共享变量。

# 一、pickling的基础

    两个模块 pickle 和 cPickle 有着相同的功能，不过 cPickle 是c实现的，可以更快：

1. >>> **import** cPickle

    你可以使用cPickle.dump来将对象序列化到文件中（save，或pickle）：

1. >>> f = file('obj.save', 'wb')
2. >>> cPickle.dump(my\_obj, f, protocol=cPickle.HIGHEST\_PROTOCOL)
3. >>> f.close()

note：如果你想要你保存的对象能够高效的存储起来，不要忘记使用cPickle.HIGHEST\_PROTOCOL. 生成的结果文件可能比默认的协议要小上20几倍。

note：用二进制模式打开你的文件 ('b') 有助于移植 (特别是在Unix和Windows之间)。

    为了反序列化 (或者 load, 或者 unpickle)一个 pickled 文件，使用 cPickle.load:

1. >>> f = file('obj.save', 'rb')
2. >>> loaded\_obj = cPickle.load(f)
3. >>> f.close()

    你可以将好几个对象pickle到同一个文件中，然后将它们都进行装载 (以相同的顺序)：

1. >>> f = file('objects.save', 'wb')
2. >>> **for** obj **in** [obj1, obj2, obj3]:
3. >>>     cPickle.dump(obj, f, protocol=cPickle.HIGHEST\_PROTOCOL)
4. >>> f.close()

然后：

1. >>> f = file('objects.save', 'rb')
2. >>> loaded\_objects = []
3. >>> **for** i **in** range(3):
4. >>>     loaded\_objects.append(cPickle.load(f))
5. >>> f.close()

更详细的 pickle用法，查看 [Python documentation](http://docs.python.org/library/pickle.html#usage).

# 二、短时间序列化

    如果你很自信的说，你序列化的类实力可以被代码的一个兼容版本所反序列化，那么pickling整个模型是一个很好的方法，例如，如果你在你程序的同一次执行中保存模型然后重新装载，或者你保存的类已经稳定了一会儿了。

    你可以控制从你的对象中什么 pickle需要被save。通过定义一个 [\_\_getstate\_\_](http://docs.python.org/library/pickle.html#object.__getstate__) 方法，和一个相似的[\_\_setstate\_\_](http://docs.python.org/library/pickle.html#object.__getstate__)方法。

    这是特别有用的，如果你的模型类包含了对一个当前使用的数据集的链接，那么你可能不想要pickle你的模型的每个实例。

    例如，你可以定义有双下划线的函数：

1. **def** \_\_getstate\_\_(self):
2. state = dict(self.\_\_dict\_\_)
3. **del** state['training\_set']
4. **return** state
6. **def** \_\_setstate\_\_(self, d):
7. self.\_\_dict\_\_.update(d)
8. self.training\_set = cPickle.load(file(self.training\_set\_file, 'rb'))

# 三、鲁棒性的序列化

    这种序列化使用了一些特定于theano的帮助函数。它使用python的pickling协议来序列化对象，不过包含在任何对象内的 ndarray 或者 CudaNdarray 对象都被一NPY文件分别独立保存了。这些NPY文件和Pickled的文件都放在一个ZIP文件内。

    这个方法的主要优势在于你升职不需要安装theano，就能看到你pickled的共享变量的值。你可以只是通过手动用numpy来装载这些参数就好。

1. numpy.load('model.zip')

    该方法当你和那些没有安装theano的人共享你的模型的时候，或者那些人使用着不同的python版本，或者你计划长时间保存你的模型，这些情况下是很受益的。(在这种情况下错误的匹配会使得unpickle对象变得很快困难）.

见 [**theano.misc.pkl\_utils.dump()**](http://deeplearning.net/software/theano/library/misc/pkl_utils.html#theano.misc.pkl_utils.dump) 和 [**theano.misc.pkl\_utils.load()**](http://deeplearning.net/software/theano/library/misc/pkl_utils.html#theano.misc.pkl_utils.load)**。**

# 四、长时间序列化

   如果你想要保存的类的实现是不稳定的，例如当函数被创建或者移除，类成员被重命名，你就应该只保存和装载你类的稳定不变的那部分。

   你可以通过定义 \_\_getstate\_\_ 和 \_\_setstate\_\_ 函数来达到目的，应该定义你想要保存的属性，而不是那些你不需要保存的。

    例如，如果这个唯一你想要保存的参数是权重矩阵W 和偏置 b,那么就可以定义：

1. **def** \_\_getstate\_\_(self):
2. **return** (self.W, self.b)
4. **def** \_\_setstate\_\_(self, state):
5. W, b = state
6. self.W = W
7. self.b = b

   如果在某个时间点上，W 重命名成了权重，而b成了偏置，那么更老的pickled文件就变得没法用了：

1. **def** \_\_getstate\_\_(self):
2. **return** (self.weights, self.bias)
4. **def** \_\_setstate\_\_(self, state):
5. W, b = state
6. self.weights = W
7. self.bias = b

    更多有关使用pickle的高级用法和它的内部信息，参考Python的 [pickle](http://docs.python.org/library/pickle.html) 文档。

参考资料：

[1] 官网：http://deeplearning.net/software/theano/tutorial/loading\_and\_saving.html

## [Theano2.1.9-基础知识之条件](http://www.cnblogs.com/shouhuxianjian/p/4590227.html)

来自：http://deeplearning.net/software/theano/tutorial/conditions.html

conditions

# 一、IfElse vs Switch

* 这两个操作都是在符号变量上建立约束条件
* IfElse 采用 boolean 条件，并且两个变量作为输入。
* Switch t采用 tensor 作为条件，也是两个变量作为输入。 switch 是一个逐元素的操作，所以比ifelse更通用。
* 然而 switch 在两个输出变量上进行评估，而 ifelse 只对一个关于条件的变量进行评估。

**Example**

1. **from** theano **import** tensor as T
2. **from** theano.ifelse **import** ifelse
3. **import** theano, time, numpy
5. a,b = T.scalars('a', 'b')
6. x,y = T.matrices('x', 'y')
8. z\_switch = T.switch(T.lt(a, b), T.mean(x), T.mean(y))
9. z\_lazy = ifelse(T.lt(a, b), T.mean(x), T.mean(y))
11. f\_switch = theano.function([a, b, x, y], z\_switch,
12. mode=theano.Mode(linker='vm'))
13. f\_lazyifelse = theano.function([a, b, x, y], z\_lazy,
14. mode=theano.Mode(linker='vm'))
16. val1 = 0.
17. val2 = 1.
18. big\_mat1 = numpy.ones((10000, 1000))
19. big\_mat2 = numpy.ones((10000, 1000))
21. n\_times = 10
23. tic = time.clock()
24. **for** i **in** xrange(n\_times):
25. f\_switch(val1, val2, big\_mat1, big\_mat2)
26. **print** 'time spent evaluating both values %f sec' % (time.clock() - tic)
28. tic = time.clock()
29. **for** i **in** xrange(n\_times):
30. f\_lazyifelse(val1, val2, big\_mat1, big\_mat2)
31. **print** 'time spent evaluating one value %f sec' % (time.clock() - tic)

    在这个例子中，fElse 操作比Switch花费更少的时间（大约节约一半时间）因为它只计算两个变量中的一个。

1. >>> python ifelse\_switch.py
2. time spent evaluating both values 0.6700 sec
3. time spent evaluating one value 0.3500 sec

    除非使用了 linker='vm' 或者 linker='cvm' ， ifelse 才会计算两个变量，然后会和switch有着一样的计算时间。尽管连接器当前设置的默认值不是cvm，不过在不久的将来会是的。

    没有自动的优化，通过使用广播的标量来替换switch成为ifelse，因为这并不见得总是更快的，见 [ticket](http://www.assembla.com/spaces/theano/tickets/764).

参考资料：

[1]官网：http://deeplearning.net/software/theano/tutorial/conditions.html

## [Theano2.1.10-基础知识之循环](http://www.cnblogs.com/shouhuxianjian/p/4590226.html)

来自：http://deeplearning.net/software/theano/tutorial/loop.html

loop

# 一、Scan

* 一个递归的通常的形式，可以用来作为循环语句。
* 约间和映射（在第一个（leading，个人翻译成第一个）维度上进行循环）是scan的特殊情况
* 沿着一些输入序列scan一个函数，然后在每个时间步上生成一个输出。
* 该函数可以查看函数的前K个时间步的结果。
* sum() 可以通过在一个列表上使用 z + x(i) 函数（初始化为Z=0）来得到结果。
* 通常来说，一个for循环可以表示成一个scan（）操作，而且scan是与theano的循环联系最紧密的。
* 使用scan而不是for循环的优势：
  + 迭代的次数是符号graph的一部分。
  + 最小化GPU的迁移（如果用到GPU的话）。
  + 通过连续的步骤计算梯度。
  + 比python中使用theano编译后的for循环稍微快一点。
  + 通过检测实际用到的内存的数量，来降低总的内存使用情况。

所有的文档可以查看库对应的： [Scan](http://deeplearning.net/software/theano/library/scan.html#lib-scan).

**1.1 Scan 例子: 逐元素计算 tanh(x(t).dot(W) + b)**

1. **import** theano
2. **import** theano.tensor as T
3. **import** numpy as np
5. # defining the tensor variables
6. X = T.matrix("X")
7. W = T.matrix("W")
8. b\_sym = T.vector("b\_sym")
10. results, updates = theano.scan(**lambda** v: T.tanh(T.dot(v, W) + b\_sym), sequences=X)
11. compute\_elementwise = theano.function(inputs=[X, W, b\_sym], outputs=[results])
13. # test values
14. x = np.eye(2, dtype=theano.config.floatX)
15. w = np.ones((2, 2), dtype=theano.config.floatX)
16. b = np.ones((2), dtype=theano.config.floatX)
17. b[1] = 2
19. **print** compute\_elementwise(x, w, b)[0]
21. # comparison with numpy
22. **print** np.tanh(x.dot(w) + b)

**1.2 Scan 例子:计算序列 x(t) = tanh(x(t - 1).dot(W) + y(t).dot(U) + p(T - t).dot(V))**

1. **import** theano
2. **import** theano.tensor as T
3. **import** numpy as np
5. # define tensor variables
6. X = T.vector("X")
7. W = T.matrix("W")
8. b\_sym = T.vector("b\_sym")
9. U = T.matrix("U")
10. Y = T.matrix("Y")
11. V = T.matrix("V")
12. P = T.matrix("P")
14. results, updates = theano.scan(**lambda** y, p, x\_tm1: T.tanh(T.dot(x\_tm1, W) + T.dot(y, U) + T.dot(p, V)),
15. sequences=[Y, P[::-1]], outputs\_info=[X])
16. compute\_seq = theano.function(inputs=[X, W, Y, U, P, V], outputs=[results])
18. # test values
19. x = np.zeros((2), dtype=theano.config.floatX)
20. x[1] = 1
21. w = np.ones((2, 2), dtype=theano.config.floatX)
22. y = np.ones((5, 2), dtype=theano.config.floatX)
23. y[0, :] = -3
24. u = np.ones((2, 2), dtype=theano.config.floatX)
25. p = np.ones((5, 2), dtype=theano.config.floatX)
26. p[0, :] = 3
27. v = np.ones((2, 2), dtype=theano.config.floatX)
29. **print** compute\_seq(x, w, y, u, p, v)[0]
31. # comparison with numpy
32. x\_res = np.zeros((5, 2), dtype=theano.config.floatX)
33. x\_res[0] = np.tanh(x.dot(w) + y[0].dot(u) + p[4].dot(v))
34. **for** i **in** range(1, 5):
35. x\_res[i] = np.tanh(x\_res[i - 1].dot(w) + y[i].dot(u) + p[4-i].dot(v))
36. **print** x\_res

**1.3 Scan 例子: 计算 X的线（指的是按照某一维度方向） 范数**

1. **import** theano
2. **import** theano.tensor as T
3. **import** numpy as np
5. # define tensor variable
6. X = T.matrix("X")
7. results, updates = theano.scan(**lambda** x\_i: T.sqrt((x\_i \*\* 2).sum()), sequences=[X])
8. compute\_norm\_lines = theano.function(inputs=[X], outputs=[results])
10. # test value
11. x = np.diag(np.arange(1, 6, dtype=theano.config.floatX), 1)
12. **print** compute\_norm\_lines(x)[0]
14. # comparison with numpy
15. **print** np.sqrt((x \*\* 2).sum(1))

**1.4 Scan 例子:计算x的列的范数**

1. **import** theano
2. **import** theano.tensor as T
3. **import** numpy as np
5. # define tensor variable
6. X = T.matrix("X")
7. results, updates = theano.scan(**lambda** x\_i: T.sqrt((x\_i \*\* 2).sum()), sequences=[X.T])
8. compute\_norm\_cols = theano.function(inputs=[X], outputs=[results])
10. # test value
11. x = np.diag(np.arange(1, 6, dtype=theano.config.floatX), 1)
12. **print** compute\_norm\_cols(x)[0]
14. # comparison with numpy
15. **print** np.sqrt((x \*\* 2).sum(0))

**1.5 Scan 例子: 计算x的迹**

1. **import** theano
2. **import** theano.tensor as T
3. **import** numpy as np
4. floatX = "float32"
6. # define tensor variable
7. X = T.matrix("X")
8. results, updates = theano.scan(**lambda** i, j, t\_f: T.cast(X[i, j] + t\_f, floatX),
9. sequences=[T.arange(X.shape[0]), T.arange(X.shape[1])],
10. outputs\_info=np.asarray(0., dtype=floatX))
11. result = results[-1]
12. compute\_trace = theano.function(inputs=[X], outputs=[result])
14. # test value
15. x = np.eye(5, dtype=theano.config.floatX)
16. x[0] = np.arange(5, dtype=theano.config.floatX)
17. **print** compute\_trace(x)[0]
19. # comparison with numpy
20. **print** np.diagonal(x).sum()

**1.6 Scan 例子:计算序列 x(t) = x(t - 2).dot(U) + x(t - 1).dot(V) + tanh(x(t - 1).dot(W) + b)**

1. **import** theano
2. **import** theano.tensor as T
3. **import** numpy as np
5. # define tensor variables
6. X = T.matrix("X")
7. W = T.matrix("W")
8. b\_sym = T.vector("b\_sym")
9. U = T.matrix("U")
10. V = T.matrix("V")
11. n\_sym = T.iscalar("n\_sym")
13. results, updates = theano.scan(**lambda** x\_tm2, x\_tm1: T.dot(x\_tm2, U) + T.dot(x\_tm1, V) + T.tanh(T.dot(x\_tm1, W) + b\_sym),
14. n\_steps=n\_sym, outputs\_info=[dict(initial=X, taps=[-2, -1])])
15. compute\_seq2 = theano.function(inputs=[X, U, V, W, b\_sym, n\_sym], outputs=[results])
17. # test values
18. x = np.zeros((2, 2), dtype=theano.config.floatX) # the initial value must be able to return x[-2]
19. x[1, 1] = 1
20. w = 0.5 \* np.ones((2, 2), dtype=theano.config.floatX)
21. u = 0.5 \* (np.ones((2, 2), dtype=theano.config.floatX) - np.eye(2, dtype=theano.config.floatX))
22. v = 0.5 \* np.ones((2, 2), dtype=theano.config.floatX)
23. n = 10
24. b = np.ones((2), dtype=theano.config.floatX)
26. **print** compute\_seq2(x, u, v, w, b, n)
28. # comparison with numpy
29. x\_res = np.zeros((10, 2))
30. x\_res[0] = x[0].dot(u) + x[1].dot(v) + np.tanh(x[1].dot(w) + b)
31. x\_res[1] = x[1].dot(u) + x\_res[0].dot(v) + np.tanh(x\_res[0].dot(w) + b)
32. x\_res[2] = x\_res[0].dot(u) + x\_res[1].dot(v) + np.tanh(x\_res[1].dot(w) + b)
33. **for** i **in** range(2, 10):
34. x\_res[i] = (x\_res[i - 2].dot(u) + x\_res[i - 1].dot(v) +
35. np.tanh(x\_res[i - 1].dot(w) + b))
36. **print** x\_res

**1.7 Scan 例子:计算 y = tanh(v.dot(A))  关于 x 的jacobian**

1. **import** theano
2. **import** theano.tensor as T
3. **import** numpy as np
5. # define tensor variables
6. v = T.vector()
7. A = T.matrix()
8. y = T.tanh(T.dot(v, A))
9. results, updates = theano.scan(**lambda** i: T.grad(y[i], v), sequences=[T.arange(y.shape[0])])
10. compute\_jac\_t = theano.function([A, v], [results], allow\_input\_downcast=True) # shape (d\_out, d\_in)
12. # test values
13. x = np.eye(5, dtype=theano.config.floatX)[0]
14. w = np.eye(5, 3, dtype=theano.config.floatX)
15. w[2] = np.ones((3), dtype=theano.config.floatX)
16. **print** compute\_jac\_t(w, x)[0]
18. # compare with numpy
19. **print** ((1 - np.tanh(x.dot(w)) \*\* 2) \* w).T

    注意到我们需要对y的索引值而不是y的元素进行迭代。原因在于scan会对它的内部函数创建一个占位符变量，该占位符变量没有和需要替换的那个变量同样的依赖条件。

**1.8 Scan 例子: 在scan中累计循环次数**

1. **import** theano
2. **import** theano.tensor as T
3. **import** numpy as np
5. # define shared variables
6. k = theano.shared(0)
7. n\_sym = T.iscalar("n\_sym")
9. results, updates = theano.scan(**lambda**:{k:(k + 1)}, n\_steps=n\_sym)
10. accumulator = theano.function([n\_sym], [], updates=updates, allow\_input\_downcast=True)
12. k.get\_value()
13. accumulator(5)
14. k.get\_value()

**1.9 Scan 例子:计算 tanh(v.dot(W) + b) \* d ，这里d 是 二项式**

1. **import** theano
2. **import** theano.tensor as T
3. **import** numpy as np
5. # define tensor variables
6. X = T.matrix("X")
7. W = T.matrix("W")
8. b\_sym = T.vector("b\_sym")
10. # define shared random stream
11. trng = T.shared\_randomstreams.RandomStreams(1234)
12. d=trng.binomial(size=W[1].shape)
14. results, updates = theano.scan(**lambda** v: T.tanh(T.dot(v, W) + b\_sym) \* d, sequences=X)
15. compute\_with\_bnoise = theano.function(inputs=[X, W, b\_sym], outputs=[results],
16. updates=updates, allow\_input\_downcast=True)
17. x = np.eye(10, 2, dtype=theano.config.floatX)
18. w = np.ones((2, 2), dtype=theano.config.floatX)
19. b = np.ones((2), dtype=theano.config.floatX)
21. **print** compute\_with\_bnoise(x, w, b)

    注意到如果你想使用一个不会通过scan循环更新的随机变量 d ，你就应该将这个变量作为参数传递给non\_sequences 。

**1.10 Scan 例子: 计算 pow(A, k)**

1. **import** theano
2. **import** theano.tensor as T
3. theano.config.warn.subtensor\_merge\_bug = False
5. k = T.iscalar("k")
6. A = T.vector("A")
8. **def** inner\_fct(prior\_result, B):
9. **return** prior\_result \* B
11. # Symbolic description of the result
12. result, updates = theano.scan(fn=inner\_fct,
13. outputs\_info=T.ones\_like(A),
14. non\_sequences=A, n\_steps=k)
16. # Scan has provided us with A \*\* 1 through A \*\* k.  Keep only the last
17. # value. Scan notices this and does not waste memory saving them.
18. final\_result = result[-1]
20. power = theano.function(inputs=[A, k], outputs=final\_result,
21. updates=updates)
23. **print** power(range(10), 2)
24. #[  0.   1.   4.   9.  16.  25.  36.  49.  64.  81.]

**1.11 Scan 例子: 计算一个多项式**

1. **import** numpy
2. **import** theano
3. **import** theano.tensor as T
4. theano.config.warn.subtensor\_merge\_bug = False
6. coefficients = theano.tensor.vector("coefficients")
7. x = T.scalar("x")
8. max\_coefficients\_supported = 10000
10. # Generate the components of the polynomial
11. full\_range=theano.tensor.arange(max\_coefficients\_supported)
12. components, updates = theano.scan(fn=**lambda** coeff, power, free\_var:
13. coeff \* (free\_var \*\* power),
14. outputs\_info=None,
15. sequences=[coefficients, full\_range],
16. non\_sequences=x)
18. polynomial = components.sum()
19. calculate\_polynomial = theano.function(inputs=[coefficients, x],
20. outputs=polynomial)
22. test\_coeff = numpy.asarray([1, 0, 2], dtype=numpy.float32)
23. **print** calculate\_polynomial(test\_coeff, 3)
24. # 19.0

# 二、Exercise

    运行两个例子。

    修改并执行多项式的例子，通过scan来进行约间。

答案（[**Solution**](http://deeplearning.net/software/theano/_downloads/loop_solution_1.py)）

1. #!/usr/bin/env python
2. # Theano tutorial
3. # Solution to Exercise in section 'Loop'
4. **from** \_\_future\_\_ **import** print\_function
5. **import** numpy
7. **import** theano
8. **import** theano.tensor as tt
10. # 1. First example
12. theano.config.warn.subtensor\_merge\_bug = False
14. k = tt.iscalar("k")
15. A = tt.vector("A")

18. **def** inner\_fct(prior\_result, A):
19. **return** prior\_result \* A
21. # Symbolic description of the result
22. result, updates = theano.scan(fn=inner\_fct,
23. outputs\_info=tt.ones\_like(A),
24. non\_sequences=A, n\_steps=k)
26. # Scan has provided us with A \*\* 1 through A \*\* k.  Keep only the last
27. # value. Scan notices this and does not waste memory saving them.
28. final\_result = result[-1]
30. power = theano.function(inputs=[A, k], outputs=final\_result,
31. updates=updates)
33. **print**(power(range(10), 2))
34. # [  0.   1.   4.   9.  16.  25.  36.  49.  64.  81.]

37. # 2. Second example
39. coefficients = tt.vector("coefficients")
40. x = tt.scalar("x")
41. max\_coefficients\_supported = 10000
43. # Generate the components of the polynomial
44. full\_range = tt.arange(max\_coefficients\_supported)
45. components, updates = theano.scan(fn=**lambda** coeff, power, free\_var:
46. coeff \* (free\_var \*\* power),
47. sequences=[coefficients, full\_range],
48. outputs\_info=None,
49. non\_sequences=x)
50. polynomial = components.sum()
51. calculate\_polynomial1 = theano.function(inputs=[coefficients, x],
52. outputs=polynomial)
54. test\_coeff = numpy.asarray([1, 0, 2], dtype=numpy.float32)
55. **print**(calculate\_polynomial1(test\_coeff, 3))
56. # 19.0
58. # 3. Reduction performed inside scan
60. theano.config.warn.subtensor\_merge\_bug = False
62. coefficients = tt.vector("coefficients")
63. x = tt.scalar("x")
64. max\_coefficients\_supported = 10000
66. # Generate the components of the polynomial
67. full\_range = tt.arange(max\_coefficients\_supported)

70. outputs\_info = tt.as\_tensor\_variable(numpy.asarray(0, 'float64'))
72. components, updates = theano.scan(fn=**lambda** coeff, power, prior\_value, free\_var:
73. prior\_value + (coeff \* (free\_var \*\* power)),
74. sequences=[coefficients, full\_range],
75. outputs\_info=outputs\_info,
76. non\_sequences=x)
78. polynomial = components[-1]
79. calculate\_polynomial = theano.function(inputs=[coefficients, x],
80. outputs=polynomial, updates=updates)
82. test\_coeff = numpy.asarray([1, 0, 2], dtype=numpy.float32)
83. **print**(calculate\_polynomial(test\_coeff, 3))
84. # 19.0

参考资料：

[1]官网：http://deeplearning.net/software/theano/tutorial/loop.html

## [Theano2.1.11-基础知识之稀疏](http://www.cnblogs.com/shouhuxianjian/p/4590225.html)

来自：http://deeplearning.net/software/theano/tutorial/sparse.html

sparse

    通常来说，稀疏矩阵可以和常规矩阵一样提供相同的功能。两者不同之处在于在内存中存储矩阵和表示矩阵的方式。 在稀疏矩阵中，只有非0元素才会被存储。这种方式带来的优点有：首先，这可以很明显的减少内存的使用‘第二；通过使用指定的稀疏算法和这种稀疏存储方法可以减少计算时间。我们通常将常规的存储矩阵叫做密集矩阵（dense matrices）。

    Theano的稀疏包提供高效的算法，不过却并不推荐在所有的矩阵和所有的情况下使用。正如一个显而易见的例子，当稀疏比例非常低的时候，就等于在常规矩阵上使用这些算法，而且本身带来的存储开销会比常规矩阵还大。稀疏比例指的是元素为0的值所占整个矩阵上所有元素的个数比。一个低稀疏比例会导致在内存上使用更多的空间，不只是需要存储实际的数据，而且还需要存储每个元素的位置信息。这同样需要更多的计算时间，然而一个密集矩阵是使用常规优化算法的，可能效果比这时候的稀疏表示矩阵更高效。其他例子可以在特定目的和矩阵的结构的联系上找到。更多的文档可以查看 [SciPy Sparse Reference](http://docs.scipy.org/doc/scipy/reference/sparse.html).

    因为稀疏矩阵不是存储在连续数组上的，所以有几种方式来存储。这通常被设计成所谓的矩阵的格式（format）。因为 Theano的稀疏矩阵包是基于SciPy 稀疏包的，完整的有关稀疏矩阵的信息可查阅 SciPy的文档。就像SciPy一样， Theano没有对那些维度不等于2的数组实现稀疏格式。

    到目前为止，Theano实现了两个稀疏矩阵的格式： csc and csr。 这些几乎是等同的，除了csc是基于矩阵的列，而csr是基于矩阵的行的。他们都是有着相同的目的：为了提供给高效的算法用来执行线性代数操作。不过缺点就是它们没法通过一个高效的方法来修改潜在矩阵的稀疏结构，例如：增加元素。这也就是说如果你在你的计算graph中会频繁的在稀疏矩阵中增加新的元素，也许一个tensor变量是更好的选择。

更多的文档可以查看 [Sparse Library Reference](http://deeplearning.net/software/theano/library/sparse/index.html#libdoc-sparse).

在进行下一步之前，来做做准备工作：

1. >>> **import** theano
2. >>> **import** numpy as np
3. >>> **import** scipy.sparse as sp
4. >>> **from** theano **import** sparse

# 一、稀疏压缩格式

    Theano支持两种稀疏压缩格式： csc 和 csr, 分别是基于列和行的。他们都有着相同的属性： data, indices, indptr 和 shape.

* data 属性是一个一维的 ndarray ，它包含稀疏矩阵所有的非0元素。
* indices 和indptr 属性是用来存储稀疏矩阵中数据的位置的。
* shape 属性，准确的说是和密集矩阵的shape属性一样的。如果从前三个属性上没法推断，那么它可以在稀疏矩阵创建的时候显式指定 。

### 1.1 应该使用哪种格式 ?

    最后，格式不会影响到data和indices属性的长度（length）。它们都是通过你所想存储的元素个数确定的。唯一会被格式改变的就是 indptr。在csc 格式下,该句子是按着列进行压缩的，所以列数量更少的时候，所需要的内存就更少。另一方面，csr 格式下，句子是按行压缩的，所以当一个矩阵有着更少行数的时候，csr格式是个更好的选择。下面就是规则的形式：

note：If shape[0] > shape[1], 使用 csr format. ；否则，使用 csc。（觉得这里和上面说的完全相反了，上面说当矩阵行少的时候用csr）。

    有时候，因为稀疏模块到现在的时间还不长，ops还没有针对两种format的操作。所以这里也是最相关的规则：

note：使用该矩阵格式可以兼容你计算graph中的ops。

关于ops和所支持的format等文档可以查阅： [Sparse Library Reference](http://deeplearning.net/software/theano/library/sparse/index.html#libdoc-sparse).

## 二、在theano中处理稀疏

    在theano中大多数的ops都是依赖于稀疏矩阵的格式（format）的。这就是为什么有两种稀疏变量的构造器： csc\_matrix 和 csr\_matrix。这些都能被普通的name和dtype参数所调用，不过没有 broadcastable 标识。这是被禁止的，因为稀疏包（和SciPy的稀疏模块一样）不提供任何方式来处理维度不等于2的情况的矩阵。稀疏矩阵的所有可接受的dtype值可以在 sparse.all\_dtypes中找到：

1. >>> sparse.all\_dtypes
2. set(['int8', 'int16', 'int32', 'int64', 'uint8', 'uint16', 'uint32', 'uint64',
3. 'float32', 'float64', 'complex64', 'complex128'])

### 2.1 转换

    为了在稀疏矩阵和密集矩阵之间相互转换。Theano提供了 dense\_from\_sparse，csr\_from\_dense 和 csc\_from\_dense 函数。不需要任何额外的细节。这里就是从稀疏到稀疏的一个完整转换的例子：

1. >>> x = sparse.csc\_matrix(name='x', dtype='float32')
2. >>> y = sparse.dense\_from\_sparse(x)
3. >>> z = sparse.csc\_from\_dense(y)

### 2.2 属性和构造

    尽管稀疏变量不允许直接访问它们的属性，不过还是可以通过 csm\_properties 来完成的。它会返回一个一维的tensor变量的元组，用来表示稀疏矩阵的内部特征。

    为了从某些属性上重构一个稀疏矩阵，可以使用函数 CSC 和 CSR 。这会以合适的格式来创建稀疏矩阵。例如，下面的代码重构一个csc矩阵到csr格式：

1. >>> x = sparse.csc\_matrix(name='x', dtype='int64')
2. >>> data, indices, indptr, shape = sparse.csm\_properties(x)
3. >>> y = sparse.CSR(data, indices, indptr, shape)
4. >>> f = theano.function([x], y)
5. >>> a = sp.csc\_matrix(np.asarray([[0, 1, 1], [0, 0, 0], [1, 0, 0]]))
6. >>> **print** a.toarray()
7. [[0 1 1]
8. [0 0 0]
9. [1 0 0]]
10. >>> **print** f(a).toarray()
11. [[0 0 1]
12. [1 0 0]
13. [1 0 0]]

        最后的例子显示，格式可以从一个转换到另一个。的确，当调用transpose函数的时候，生成的矩阵的稀疏特征就不会和输入的时候一样了

**2.3 结构（structured）操作**

        有几个ops是用在稀疏矩阵的非常奇特的结构上的。这些ops是结构式的（structured），而且不会在稀疏矩阵的0元素上有任何的计算操作。 它们被认为是只用在后面（暂时不知道是指哪个）的数据属性上的。 注意，这些结构操作都提供一个结构梯度（structured gradient）。更多的解释如下：

1. >>> x = sparse.csc\_matrix(name='x', dtype='float32')
2. >>> y = sparse.structured\_add(x, 2)
3. >>> f = theano.function([x], y)
4. >>> a = sp.csc\_matrix(np.asarray([[0, 0, -1], [0, -2, 1], [3, 0, 0]], dtype='float32'))
5. >>> **print** a.toarray()
6. [[ 0.  0. -1.]
7. [ 0. -2.  1.]
8. [ 3.  0.  0.]]
9. >>> **print** f(a).toarray()
10. [[ 0.  0.  1.]
11. [ 0.  0.  3.]
12. [ 5.  0.  0.]]

### 2.4 梯度

     在稀疏模块中ops的梯度同样可以是structured。一些ops提供一个 flag 来指定该梯度是否structured。该文档可以用来决定一个op的梯度是否是常规的还是structured还是它的实现是否能够被修改。相似于structured ops，当计算了一个structured gradient的时候，该计算只会应用在系数矩阵的非0元素上。

    更多有关特定op上的梯度可以查阅： [Sparse Library Reference](http://deeplearning.net/software/theano/library/sparse/index.html#libdoc-sparse).

参考资料：

[1]官网：http://deeplearning.net/software/theano/tutorial/sparse.html

## [Theano2.1.12-基础知识之使用GPU](http://www.cnblogs.com/shouhuxianjian/p/4590224.html)

来自：http://deeplearning.net/software/theano/tutorial/using\_gpu.html

using the GPU

    想要看GPU的介绍性的讨论和对密集并行计算的使用，查阅：[GPGPU](http://en.wikipedia.org/wiki/GPGPU).

    theano设计的一个目标就是在一个抽象层面上进行特定的计算，所以内部的函数编译器需要灵活的处理这些计算，其中一个灵活性体现在可以在显卡上进行计算。

    当前有两种方式来使用gpu，一种只支持NVIDIA cards ([CUDA backend](http://deeplearning.net/software/theano/tutorial/using_gpu.html#cuda)) ；另一种，还在开发中，可以支持任何 OpenCL设备，就像和NVIDIA cards ([GpuArray Backend](http://deeplearning.net/software/theano/tutorial/using_gpu.html#gpuarray))一样。

# 一、CUDA backend

    如果你没有准备好，那么就需要安装Nvidia 的 GPU编程工具链 (CUDA)，然后配置好 Theano。我们提供了安装指南[Linux](http://deeplearning.net/software/theano/install.html#gpu-linux), [MacOS](http://deeplearning.net/software/theano/install.html#gpu-macos) and [*Windows*](http://deeplearning.net/software/theano/install_windows.html#gpu-windows).（[我的安装](http://blog.csdn.net/shouhuxianjian/article/details/46004159)）。

### 1.1 测试theano和GPU

    为了检查你的GPU是否启用了，可以剪切下面的代码然后保存成一个文件，运行看看。

1. **from** theano **import** function, config, shared, sandbox
2. **import** theano.tensor as T
3. **import** numpy
4. **import** time
6. vlen = 10 \* 30 \* 768  # 10 x #cores x # threads per core
7. iters = 1000
9. rng = numpy.random.RandomState(22)
10. x = shared(numpy.asarray(rng.rand(vlen), config.floatX))
11. f = function([], T.exp(x))
12. **print** f.maker.fgraph.toposort()
13. t0 = time.time()
14. **for** i **in** xrange(iters):
15. r = f()
16. t1 = time.time()
17. **print** 'Looping %d times took' % iters, t1 - t0, 'seconds'
18. **print** 'Result is', r
19. **if** numpy.any([isinstance(x.op, T.Elemwise) **for** x **in** f.maker.fgraph.toposort()]):
20. **print** 'Used the cpu'
21. **else**:
22. **print** 'Used the gpu'

    该程序会计算一堆随机数的exp() 。注意到我们使用了 shared 函数来确保输入的x 是存储在显卡设备上的。

    如果运行该程序(保存文件名为check1.py)，而且device=cpu, 那么计算机将会花费大约 3 ；而在GPU 上，只需要0.64秒。不过 GPU不会一直生成完全和CPU一致的浮点数。 作为一个基准来说，调用numpy.exp(x.get\_value()) 的一个循环会花费大约 46秒。

1. $ THEANO\_FLAGS=mode=FAST\_RUN,device=cpu,floatX=float32 python check1.py
2. [Elemwise{exp,no\_inplace}(<TensorType(float32, vector)>)]
3. Looping 1000 times took 3.06635117531 seconds
4. Result **is** [ 1.23178029  1.61879337  1.52278066 ...,  2.20771813  2.29967761
5. 1.62323284]
6. Used the cpu
8. $ THEANO\_FLAGS=mode=FAST\_RUN,device=gpu,floatX=float32 python check1.py
9. Using gpu device 0: GeForce GTX 580
10. [GpuElemwise{exp,no\_inplace}(<CudaNdarrayType(float32, vector)>), HostFromGpu(GpuElemwise{exp,no\_inplace}.0)]
11. Looping 1000 times took 0.638810873032 seconds
12. Result **is** [ 1.23178029  1.61879349  1.52278066 ...,  2.20771813  2.29967761
13. 1.62323296]
14. Used the gpu

    注意到在theano中GPU的操作在目前来说，只支持 floatX 为float32类型。

### 1.2 返回设备分配数据的句柄

    在前面的例子中，加速并没有那么明显，这是因为函数返回的结果是作为一个 NumPy ndarray，而为了方便，已经从设备复制到主机上了。这就是为什么在device=gpu下很容易交换的原因，不过如果你不建议更少的可移植性，可以通过改变graph来用GPU的存储结果表示一个计算的过程来得到更大的加速。 gpu\_from\_host 操作也就是说“将输入从主机复制到GPU上”，然后在T.exp(x)被GPU版本的exp()替换后进行优化。

1. **from** theano **import** function, config, shared, sandbox
2. **import** theano.sandbox.cuda.basic\_ops
3. **import** theano.tensor as T
4. **import** numpy
5. **import** time
7. vlen = 10 \* 30 \* 768  # 10 x #cores x # threads per core
8. iters = 1000
10. rng = numpy.random.RandomState(22)
11. x = shared(numpy.asarray(rng.rand(vlen), config.floatX))
12. f = function([], sandbox.cuda.basic\_ops.gpu\_from\_host(T.exp(x)))
13. **print** f.maker.fgraph.toposort()
14. t0 = time.time()
15. **for** i **in** xrange(iters):
16. r = f()
17. t1 = time.time()
18. **print** 'Looping %d times took' % iters, t1 - t0, 'seconds'
19. **print** 'Result is', r
20. **print** 'Numpy result is', numpy.asarray(r)
21. **if** numpy.any([isinstance(x.op, T.Elemwise) **for** x **in** f.maker.fgraph.toposort()]):
22. **print** 'Used the cpu'
23. **else**:
24. **print** 'Used the gpu'

输出结果为：

1. $ THEANO\_FLAGS=mode=FAST\_RUN,device=gpu,floatX=float32 python check2.py
2. Using gpu device 0: GeForce GTX 580
3. [GpuElemwise{exp,no\_inplace}(<CudaNdarrayType(float32, vector)>)]
4. Looping 1000 times took 0.34898686409 seconds
5. Result **is** <CudaNdarray object at 0x6a7a5f0>
6. Numpy result **is** [ 1.23178029  1.61879349  1.52278066 ...,  2.20771813  2.29967761
7. 1.62323296]
8. Used the gpu

    这里我们通过简单的不要将结果数组复制回主机的方式省掉了大约50%的运行时间。通过每次的函数调用返回的对象不是一个NumPy array，而是一个 “CudaNdarray”，后者可以通过正常的Numpy casting机制（例如numpy.asarray()）来转换成一个NumPy ndarray。

    对更对你可以使用borrow flag加速的资料，查阅：[Borrowing when Constructing Function Objects](http://deeplearning.net/software/theano/tutorial/aliasing.html#borrowfunction).

### 1.3 在GPU上加速的是什么？

     在当我们接着优化我们的实现的时候，效果的特性也会改变，而且在从设备到设备之间会有所变化，不过现在还是给出一个粗略的想法吧：

* 只有float32 的数据类型的计算可以加速。针对float64的更好的支持期待将来的硬件，不过在目前（2010年1月）float64还是相当慢的。
* 当参数是足够大而保持30个处理器都工作的时候，矩阵乘法，卷积和大型的逐元素计算可以加速大概5-50x。
* 索引、维度重排和常量时间的reshaping在gpu和cpu上一样块。
* 在张量上基于行/列的求和在gpu上可能会比cpu上慢一点。
* 设备与主机之间大量的数据的复制是相当慢的，通常会抵消掉在数据上一两个加速函数的大部分优势。让gpu取得性能上的提升的关键取决于数据传输到设备上的时间消耗。

### 1.4 在gpu上提升效果的提示

* 考虑将floatX=float32 加到你的 .theanorc 文件中。
* 使用theano flag allow\_gc=False. 见 [GPU Async capabilities](http://deeplearning.net/software/theano/tutorial/using_gpu.html#gpu-async)
* 推荐使用构造器，如matrix, vector 和 scalar 来替换dmatrix, dvector 和 dscalar。因为前者当设定floatX = float32 的时候回使用float32类型的变量。
* 确保你的输出变量为float32 dtype而不是float64。在graph中更多的float32变量会让你将更多的工作放在gpu上实现。
* 使用shared float32变量存储频繁访问的数据（见shared（））来最大程度的减少转移到gpu设备上花费的时间。当使用gpu的时候，float32 张量共享变量存储在gpu上，并默认的使用这些变量来消除到gpu上的传输时间。（这里的意思应该是创建的时候就放在gpu上，而无需每次调用都从cpu上传给gpu，从而这份数据能够一直保持在gpu上，减少多次的传输）。
* 如果你对你得到的效果不满意，试着用 mode='ProfileMode'来建立你的函数。这在程序终止的时候，会打印出一些时间信息。如果一个op或者apply花费了它共享还多的时间，那么如果你知道一些gpu变成，就可以看看在theano.sandbox.cuda上它是怎么实现的。检查下载cpu上花费的时间比例Xs(X%)，和在gpu上花费的时间比例 Xs(X%)和在传输操作上花费的时间比例Xs(X%) 。这可以告诉你你的graph所花费的时间是在gpu上还是更多的在内存的传输上。
* 使用 nvcc 选项。 nvcc 支持一些选项来加速某些计算： -ftz=true to [flush denormals values to zeros.](https://developer.nvidia.com/content/cuda-pro-tip-flush-denormals-confidence), –prec-div=false 和 –prec-sqrt=false 选项可以通过使用更少的精度来对除法和平方根操作进行加速，。你可以通过  nvcc.flags=–use\_fast\_math Theano flag 来一次启用它们，或者如子nvcc.flags=-ftz=true –prec-div=false一样分别对它们进行启用。

### 1.5 GPU 异步功能

     从Theano 0.6开始，我们就开始使用gpu的异步功能了。这可以让我们运行的更快，不过可能会让一些错误在它们本应该出现的地方延迟抛出异常。则会导致当分析 theano apply节点的时候有些困难。这里有一个 NVIDIA 驱动特性有助于解决这些问题。如果你将环境变量设置成CUDA\_LAUNCH\_BLOCKING=1 那么，所有的kernel调用都会自动同步的。这会降低性能，不过却提供很好的profiling和合理的位置错误信息。

   该特性会与theano的中间结果的垃圾回收相关联。为了获取该特性的大部分效果，你需要禁用gc来在graph中插入同步点。设置theano flag allow\_gc=False 来得到甚至更快的速度！不过这会引起内存使用率上升的问题。

### 1.6 改变共享变量的值

    为了改变共享变量的值，即对进程提供新的数据，可以使用函数shared\_variable.set\_value(new\_value). 更详细的资料，查阅 [Understanding Memory Aliasing for Speed and Correctness](http://deeplearning.net/software/theano/tutorial/aliasing.html#aliasing).

#### 练习：再次拿逻辑回归做例子

1. **import** numpy
2. **import** theano
3. **import** theano.tensor as T
4. rng = numpy.random
6. N = 400
7. feats = 784
8. D = (rng.randn(N, feats).astype(theano.config.floatX),
9. rng.randint(size=N,low=0, high=2).astype(theano.config.floatX))
10. training\_steps = 10000
12. # Declare Theano symbolic variables
13. x = T.matrix("x")
14. y = T.vector("y")
15. w = theano.shared(rng.randn(feats).astype(theano.config.floatX), name="w")
16. b = theano.shared(numpy.asarray(0., dtype=theano.config.floatX), name="b")
17. x.tag.test\_value = D[0]
18. y.tag.test\_value = D[1]
19. #print "Initial model:"
20. #print w.get\_value(), b.get\_value()
22. # Construct Theano expression graph
23. p\_1 = 1 / (1 + T.exp(-T.dot(x, w)-b)) # Probability of having a one
24. prediction = p\_1 > 0.5 # The prediction that is done: 0 or 1
25. xent = -y\*T.log(p\_1) - (1-y)\*T.log(1-p\_1) # Cross-entropy
26. cost = xent.mean() + 0.01\*(w\*\*2).sum() # The cost to optimize
27. gw,gb = T.grad(cost, [w,b])
29. # Compile expressions to functions
30. train = theano.function(
31. inputs=[x,y],
32. outputs=[prediction, xent],
33. updates={w:w-0.01\*gw, b:b-0.01\*gb},
34. name = "train")
35. predict = theano.function(inputs=[x], outputs=prediction,
36. name = "predict")
38. **if** any([x.op.\_\_class\_\_.\_\_name\_\_ **in** ['Gemv', 'CGemv', 'Gemm', 'CGemm'] **for** x **in**
39. train.maker.fgraph.toposort()]):
40. **print** 'Used the cpu'
41. **elif** any([x.op.\_\_class\_\_.\_\_name\_\_ **in** ['GpuGemm', 'GpuGemv'] **for** x **in**
42. train.maker.fgraph.toposort()]):
43. **print** 'Used the gpu'
44. **else**:
45. **print** 'ERROR, not able to tell if theano used the cpu or the gpu'
46. **print** train.maker.fgraph.toposort()
48. **for** i **in** range(training\_steps):
49. pred, err = train(D[0], D[1])
50. #print "Final model:"
51. #print w.get\_value(), b.get\_value()
53. **print** "target values for D"
54. **print** D[1]
56. **print** "prediction on D"
57. **print** predict(D[0])

    修改并通过使用floatX= float32来在gpu上执行该例子，并使用time python file.py。来查看执行时间 (帮助资料：[Configuration Settings and Compiling Mode](http://deeplearning.net/software/theano/tutorial/modes.html#using-modes))。

从cpu到gpu上有速度的提升吗？

Where does it come from? (Use ProfileMode)

在gpu上如何有更好的速度的提升？

note：

* 当前只支持32 位 floats (其他待开发)。
* 有着float32 dtype的Shared 变量默认会放到gpu内存空间上.
* 当前一个gpu被限制成只允许一个进程。
* 使用Theano flag device=gpu 来请求使用gpu设备。
* 当你有多个gpu的时候，使用 device=gpu{0, 1, ...} 来指定具体的那个。
* 在代码中使用Theano flag floatX=float32 (through theano.config.floatX) 。
* 在存储到一个shared变量之前记得Cast 输入。
* 避免本该cast到float32的int32 自动变成float64：
  + 在代码中手动插入cast或者使用 [u]int{8,16}.
  + 在均值操作的周围手动插入cast (这会涉及到length的除法，而这是一个int64类型的).
  + 注意：一个新的casting机制在开发中。

答案（[**Solution**](http://deeplearning.net/software/theano/_downloads/using_gpu_solution_1.py)）

1. #!/usr/bin/env python
2. # Theano tutorial
3. # Solution to Exercise in section 'Using the GPU'

6. # 1. Raw results

9. **from** \_\_future\_\_ **import** print\_function
10. **import** numpy
11. **import** theano
12. **import** theano.tensor as tt
14. **from** theano **import** sandbox, Out
16. theano.config.floatX = 'float32'
18. rng = numpy.random
20. N = 400
21. feats = 784
22. D = (rng.randn(N, feats).astype(theano.config.floatX),
23. rng.randint(size=N, low=0, high=2).astype(theano.config.floatX))
24. training\_steps = 10000
26. # Declare Theano symbolic variables
27. x = theano.shared(D[0], name="x")
28. y = theano.shared(D[1], name="y")
29. w = theano.shared(rng.randn(feats).astype(theano.config.floatX), name="w")
30. b = theano.shared(numpy.asarray(0., dtype=theano.config.floatX), name="b")
31. x.tag.test\_value = D[0]
32. y.tag.test\_value = D[1]
33. #print "Initial model:"
34. #print w.get\_value(), b.get\_value()
36. # Construct Theano expression graph
37. p\_1 = 1 / (1 + tt.exp(-tt.dot(x, w) - b))  # Probability of having a one
38. prediction = p\_1 > 0.5  # The prediction that is done: 0 or 1
39. xent = -y \* tt.log(p\_1) - (1 - y) \* tt.log(1 - p\_1)  # Cross-entropy
40. cost = tt.cast(xent.mean(), 'float32') + \
41. 0.01 \* (w \*\* 2).sum()  # The cost to optimize
42. gw, gb = tt.grad(cost, [w, b])
44. """
45. # Compile expressions to functions
46. train = theano.function(
47. inputs=[x, y],
48. outputs=[Out(theano.sandbox.cuda.basic\_ops.gpu\_from\_host(tt.cast(prediction, 'float32')),borrow=True), Out(theano.sandbox.cuda.basic\_ops.gpu\_from\_host(tt.cast(xent, 'float32')), borrow=True)],
49. updates={w: w - 0.01 \* gw, b: b - 0.01 \* gb},
50. name="train")
51. predict = theano.function(inputs=[x], outputs=Out(theano.sandbox.cuda.basic\_ops.gpu\_from\_host(tt.cast(prediction, 'float32')), borrow=True),
52. name="predict")
53. """
55. # Compile expressions to functions
56. train = theano.function(
57. inputs=[],
58. outputs=[prediction, xent],
59. updates={w: w - 0.01 \* gw, b: b - 0.01 \* gb},
60. name="train")
61. predict = theano.function(inputs=[], outputs=prediction,
62. name="predict")
64. **if** any([x.op.\_\_class\_\_.\_\_name\_\_ **in** ['Gemv', 'CGemv', 'Gemm', 'CGemm'] **for** x **in**
65. train.maker.fgraph.toposort()]):
66. **print**('Used the cpu')
67. **elif** any([x.op.\_\_class\_\_.\_\_name\_\_ **in** ['GpuGemm', 'GpuGemv'] **for** x **in**
68. train.maker.fgraph.toposort()]):
69. **print**('Used the gpu')
70. **else**:
71. **print**('ERROR, not able to tell if theano used the cpu or the gpu')
72. **print**(train.maker.fgraph.toposort())
74. **for** i **in** range(training\_steps):
75. pred, err = train()
76. #print "Final model:"
77. #print w.get\_value(), b.get\_value()
79. **print**("target values for D")
80. **print**(D[1])
82. **print**("prediction on D")
83. **print**(predict())
85. """
87. # 2. Profiling

90. # 2.1 Profiling for CPU computations
92. # In your terminal, type:
93. $ THEANO\_FLAGS=profile=True,device=cpu python using\_gpu\_solution\_1.py
95. # You'll see first the output of the script:
96. Used the cpu
97. target values for D
98. prediction on D
100. # Followed by the output of profiling.. You'll see profiling results for each function
101. # in the script, followed by a summary for all functions.
102. # We'll show here only the summary:
104. Results were produced using an Intel(R) Core(TM) i7-4820K CPU @ 3.70GHz
106. Function profiling
107. ==================
108. Message: Sum of all(3) printed profiles at exit excluding Scan op profile.
109. Time in 10002 calls to Function.\_\_call\_\_: 1.590916e+00s
110. Time in Function.fn.\_\_call\_\_: 1.492365e+00s (93.805%)
111. Time in thunks: 1.408159e+00s (88.512%)
112. Total compile time: 6.309664e+00s
113. Number of Apply nodes: 25
114. Theano Optimizer time: 4.848340e-01s
115. Theano validate time: 5.454302e-03s
116. Theano Linker time (includes C, CUDA code generation/compiling): 5.691789e+00s
118. Class
119. ---
120. <% time> <sum %> <apply time> <time per call> <type> <#call> <#apply> <Class name>
121. 59.6%    59.6%       0.839s       4.19e-05s     C    20001       3   theano.tensor.blas\_c.CGemv
122. 30.1%    89.7%       0.424s       4.71e-06s     C    90001      10   theano.tensor.elemwise.Elemwise
123. 5.5%    95.2%       0.078s       7.79e-02s     Py       1       1   theano.tensor.blas.Gemv
124. 1.9%    97.1%       0.026s       1.30e-06s     C    20001       3   theano.tensor.basic.Alloc
125. 1.3%    98.4%       0.018s       1.85e-06s     C    10000       1   theano.tensor.elemwise.Sum
126. 1.0%    99.4%       0.014s       4.78e-07s     C    30001       4   theano.tensor.elemwise.DimShuffle
127. 0.6%   100.0%       0.008s       4.23e-07s     C    20001       3   theano.compile.ops.Shape\_i
128. ... (remaining 0 Classes account for   0.00%(0.00s) of the runtime)
130. Ops
131. ---
132. <% time> <sum %> <apply time> <time per call> <type> <#call> <#apply> <Op name>
133. 59.6%    59.6%       0.839s       4.19e-05s     C     20001        3   CGemv{inplace}
134. 15.8%    75.4%       0.223s       2.23e-05s     C     10000        1   Elemwise{Composite{[sub(mul(i0, scalar\_softplus(i1)), mul(i2, i3, scalar\_softplus(i4)))]}}[(0, 4)]
135. 7.7%    83.1%       0.109s       1.09e-05s     C     10000        1   Elemwise{Composite{[add(mul(scalar\_sigmoid(i0), i1, i2, i3), true\_div(mul(scalar\_sigmoid(neg(i0)), i4), i5))]}}[(0, 0)]
136. 5.5%    88.7%       0.078s       7.79e-02s     Py       1        1   Gemv{no\_inplace}
137. 4.3%    92.9%       0.060s       6.00e-06s     C     10000        1   Elemwise{Composite{[GT(scalar\_sigmoid(i0), i1)]}}
138. 1.9%    94.8%       0.026s       1.30e-06s     C     20001        3   Alloc
139. 1.3%    96.1%       0.018s       1.85e-06s     C     10000        1   Sum{acc\_dtype=float64}
140. 0.7%    96.8%       0.009s       4.73e-07s     C     20001        3   InplaceDimShuffle{x}
141. 0.6%    97.4%       0.009s       8.52e-07s     C     10000        1   Elemwise{sub,no\_inplace}
142. 0.6%    98.0%       0.008s       4.23e-07s     C     20001        3   Shape\_i{0}
143. 0.5%    98.5%       0.007s       7.06e-07s     C     10000        1   Elemwise{Composite{[sub(neg(i0), i1)]}}[(0, 0)]
144. 0.5%    98.9%       0.007s       6.57e-07s     C     10000        1   Elemwise{neg,no\_inplace}
145. 0.3%    99.3%       0.005s       4.88e-07s     C     10000        1   InplaceDimShuffle{1,0}
146. 0.3%    99.5%       0.004s       3.78e-07s     C     10000        1   Elemwise{inv,no\_inplace}
147. 0.2%    99.8%       0.003s       3.44e-07s     C     10000        1   Elemwise{Cast{float32}}
148. 0.2%   100.0%       0.003s       3.01e-07s     C     10000        1   Elemwise{Composite{[sub(i0, mul(i1, i2))]}}[(0, 0)]
149. 0.0%   100.0%       0.000s       8.11e-06s     C        1        1   Elemwise{Composite{[GT(scalar\_sigmoid(neg(sub(neg(i0), i1))), i2)]}}
150. ... (remaining 0 Ops account for   0.00%(0.00s) of the runtime)
152. Apply
153. ------
154. <% time> <sum %> <apply time> <time per call> <#call> <id> <Apply name>
155. 31.6%    31.6%       0.445s       4.45e-05s   10000     7   CGemv{inplace}(Alloc.0, TensorConstant{1.0}, x, w, TensorConstant{0.0})
156. 27.9%    59.6%       0.393s       3.93e-05s   10000    17   CGemv{inplace}(w, TensorConstant{-0.00999999977648}, x.T, Elemwise{Composite{[add(mul(scalar\_sigmoid(i0), i1, i2, i3), true\_div(mul(scalar\_sigmoid(neg(i0)), i4), i5))]}}[(0, 0)].0, TensorConstant{0.999800026417})
157. 15.8%    75.4%       0.223s       2.23e-05s   10000    14   Elemwise{Composite{[sub(mul(i0, scalar\_softplus(i1)), mul(i2, i3, scalar\_softplus(i4)))]}}[(0, 4)](y, Elemwise{Composite{[sub(neg(i0), i1)]}}[(0, 0)].0, TensorConstant{(1,) of -1.0}, Elemwise{sub,no\_inplace}.0, Elemwise{neg,no\_inplace}.0)
158. 7.7%    83.1%       0.109s       1.09e-05s   10000    15   Elemwise{Composite{[add(mul(scalar\_sigmoid(i0), i1, i2, i3), true\_div(mul(scalar\_sigmoid(neg(i0)), i4), i5))]}}[(0, 0)](Elemwise{Composite{[sub(neg(i0), i1)]}}[(0, 0)].0, TensorConstant{(1,) of -1.0}, Alloc.0, y, Elemwise{sub,no\_inplace}.0, Elemwise{Cast{float32}}.0)
159. 5.5%    88.7%       0.078s       7.79e-02s      1     0   Gemv{no\_inplace}(aa, TensorConstant{1.0}, xx, yy, TensorConstant{0.0})
160. 4.3%    92.9%       0.060s       6.00e-06s   10000    13   Elemwise{Composite{[GT(scalar\_sigmoid(i0), i1)]}}(Elemwise{neg,no\_inplace}.0, TensorConstant{(1,) of 0.5})
161. 1.3%    94.2%       0.018s       1.85e-06s   10000    16   Sum{acc\_dtype=float64}(Elemwise{Composite{[add(mul(scalar\_sigmoid(i0), i1, i2, i3), true\_div(mul(scalar\_sigmoid(neg(i0)), i4), i5))]}}[(0, 0)].0)
162. 1.0%    95.2%       0.013s       1.34e-06s   10000     5   Alloc(TensorConstant{0.0}, Shape\_i{0}.0)
163. 0.9%    96.1%       0.013s       1.27e-06s   10000    12   Alloc(Elemwise{inv,no\_inplace}.0, Shape\_i{0}.0)
164. 0.6%    96.7%       0.009s       8.52e-07s   10000     4   Elemwise{sub,no\_inplace}(TensorConstant{(1,) of 1.0}, y)
165. 0.5%    97.2%       0.007s       7.06e-07s   10000     9   Elemwise{Composite{[sub(neg(i0), i1)]}}[(0, 0)](CGemv{inplace}.0, InplaceDimShuffle{x}.0)
166. 0.5%    97.6%       0.007s       6.57e-07s   10000    11   Elemwise{neg,no\_inplace}(Elemwise{Composite{[sub(neg(i0), i1)]}}[(0, 0)].0)
167. 0.4%    98.1%       0.006s       6.27e-07s   10000     0   InplaceDimShuffle{x}(b)
168. 0.4%    98.5%       0.006s       5.90e-07s   10000     1   Shape\_i{0}(x)
169. 0.3%    98.9%       0.005s       4.88e-07s   10000     2   InplaceDimShuffle{1,0}(x)
170. 0.3%    99.1%       0.004s       3.78e-07s   10000    10   Elemwise{inv,no\_inplace}(Elemwise{Cast{float32}}.0)
171. 0.2%    99.4%       0.003s       3.44e-07s   10000     8   Elemwise{Cast{float32}}(InplaceDimShuffle{x}.0)
172. 0.2%    99.6%       0.003s       3.19e-07s   10000     6   InplaceDimShuffle{x}(Shape\_i{0}.0)
173. 0.2%    99.8%       0.003s       3.01e-07s   10000    18   Elemwise{Composite{[sub(i0, mul(i1, i2))]}}[(0, 0)](b, TensorConstant{0.00999999977648}, Sum{acc\_dtype=float64}.0)
174. 0.2%   100.0%       0.003s       2.56e-07s   10000     3   Shape\_i{0}(y)
175. ... (remaining 5 Apply instances account for 0.00%(0.00s) of the runtime)


179. # 2.2 Profiling for GPU computations
181. # In your terminal, type:
182. $ CUDA\_LAUNCH\_BLOCKING=1 THEANO\_FLAGS=profile=True,device=gpu python using\_gpu\_solution\_1.py
184. # You'll see first the output of the script:
185. Used the gpu
186. target values for D
187. prediction on D
189. Results were produced using a GeForce GTX TITAN
191. # Profiling summary for all functions:
193. Function profiling
194. ==================
195. Message: Sum of all(3) printed profiles at exit excluding Scan op profile.
196. Time in 10002 calls to Function.\_\_call\_\_: 3.535239e+00s
197. Time in Function.fn.\_\_call\_\_: 3.420863e+00s (96.765%)
198. Time in thunks: 2.865905e+00s (81.067%)
199. Total compile time: 4.728150e-01s
200. Number of Apply nodes: 36
201. Theano Optimizer time: 4.283385e-01s
202. Theano validate time: 7.687330e-03s
203. Theano Linker time (includes C, CUDA code generation/compiling): 2.801418e-02s
205. Class
206. ---
207. <% time> <sum %> <apply time> <time per call> <type> <#call> <#apply> <Class name>
208. 45.7%    45.7%       1.308s       1.64e-05s     C    80001       9   theano.sandbox.cuda.basic\_ops.GpuElemwise
209. 17.2%    62.8%       0.492s       2.46e-05s     C    20002       4   theano.sandbox.cuda.blas.GpuGemv
210. 15.1%    77.9%       0.433s       2.17e-05s     C    20001       3   theano.sandbox.cuda.basic\_ops.GpuAlloc
211. 8.2%    86.1%       0.234s       1.17e-05s     C    20002       4   theano.sandbox.cuda.basic\_ops.HostFromGpu
212. 7.2%    93.3%       0.207s       2.07e-05s     C    10000       1   theano.sandbox.cuda.basic\_ops.GpuCAReduce
213. 4.4%    97.7%       0.127s       1.27e-05s     C    10003       4   theano.sandbox.cuda.basic\_ops.GpuFromHost
214. 0.9%    98.6%       0.025s       8.23e-07s     C    30001       4   theano.sandbox.cuda.basic\_ops.GpuDimShuffle
215. 0.7%    99.3%       0.020s       9.88e-07s     C    20001       3   theano.tensor.elemwise.Elemwise
216. 0.5%    99.8%       0.014s       7.18e-07s     C    20001       3   theano.compile.ops.Shape\_i
217. 0.2%   100.0%       0.006s       5.78e-07s     C    10000       1   theano.tensor.elemwise.DimShuffle
218. ... (remaining 0 Classes account for   0.00%(0.00s) of the runtime)
220. Ops
221. ---
222. <% time> <sum %> <apply time> <time per call> <type> <#call> <#apply> <Op name>
223. 17.2%    17.2%       0.492s       2.46e-05s     C     20001        3   GpuGemv{inplace}
224. 8.2%    25.3%       0.234s       1.17e-05s     C     20002        4   HostFromGpu
225. 8.0%    33.3%       0.228s       2.28e-05s     C     10001        2   GpuAlloc{memset\_0=True}
226. 7.4%    40.7%       0.211s       2.11e-05s     C     10000        1   GpuElemwise{Composite{[sub(mul(i0, scalar\_softplus(i1)), mul(i2, i3, scalar\_softplus(i4)))]},no\_inplace}
227. 7.2%    47.9%       0.207s       2.07e-05s     C     10000        1   GpuCAReduce{add}{1}
228. 7.1%    55.0%       0.205s       2.05e-05s     C     10000        1   GpuAlloc
229. 6.9%    62.0%       0.198s       1.98e-05s     C     10000        1   GpuElemwise{sub,no\_inplace}
230. 6.9%    68.9%       0.198s       1.98e-05s     C     10000        1   GpuElemwise{inv,no\_inplace}
231. 6.2%    75.1%       0.178s       1.78e-05s     C     10000        1   GpuElemwise{neg,no\_inplace}
232. 5.6%    80.6%       0.159s       1.59e-05s     C     10000        1   GpuElemwise{Composite{[add(mul(scalar\_sigmoid(i0), i1, i2, i3), true\_div(mul(i4, i5), i6))]}}[(0, 0)]
233. 4.4%    85.1%       0.127s       1.27e-05s     C     10003        4   GpuFromHost
234. 4.3%    89.4%       0.124s       1.24e-05s     C     10000        1   GpuElemwise{Composite{[sub(neg(i0), i1)]}}[(0, 0)]
235. 4.2%    93.6%       0.121s       1.21e-05s     C     10000        1   GpuElemwise{ScalarSigmoid}[(0, 0)]
236. 4.2%    97.7%       0.119s       1.19e-05s     C     10000        1   GpuElemwise{Composite{[sub(i0, mul(i1, i2))]}}[(0, 0)]
237. 0.5%    98.2%       0.014s       7.18e-07s     C     20001        3   Shape\_i{0}
238. 0.5%    98.7%       0.013s       1.33e-06s     C     10001        2   Elemwise{gt,no\_inplace}
239. 0.3%    99.0%       0.010s       9.81e-07s     C     10000        1   GpuDimShuffle{1,0}
240. 0.3%    99.3%       0.008s       7.90e-07s     C     10000        1   GpuDimShuffle{0}
241. 0.2%    99.6%       0.007s       6.97e-07s     C     10001        2   GpuDimShuffle{x}
242. 0.2%    99.8%       0.006s       6.50e-07s     C     10000        1   Elemwise{Cast{float32}}
243. ... (remaining 3 Ops account for   0.20%(0.01s) of the runtime)
245. Apply
246. ------
247. <% time> <sum %> <apply time> <time per call> <#call> <id> <Apply name>
248. 8.8%     8.8%       0.251s       2.51e-05s   10000    22   GpuGemv{inplace}(w, TensorConstant{-0.00999999977648}, GpuDimShuffle{1,0}.0, GpuElemwise{Composite{[add(mul(scalar\_sigmoid(i0), i1, i2, i3), true\_div(mul(i4, i5), i6))]}}[(0, 0)].0, TensorConstant{0.999800026417})
249. 8.4%    17.2%       0.241s       2.41e-05s   10000     7   GpuGemv{inplace}(GpuAlloc{memset\_0=True}.0, TensorConstant{1.0}, x, w, TensorConstant{0.0})
250. 8.0%    25.1%       0.228s       2.28e-05s   10000     5   GpuAlloc{memset\_0=True}(CudaNdarrayConstant{[ 0.]}, Shape\_i{0}.0)
251. 7.4%    32.5%       0.211s       2.11e-05s   10000    13   GpuElemwise{Composite{[sub(mul(i0, scalar\_softplus(i1)), mul(i2, i3, scalar\_softplus(i4)))]},no\_inplace}(y, GpuElemwise{Composite{[sub(neg(i0), i1)]}}[(0, 0)].0, CudaNdarrayConstant{[-1.]}, GpuElemwise{sub,no\_inplace}.0, GpuElemwise{neg,no\_inplace}.0)
252. 7.2%    39.7%       0.207s       2.07e-05s   10000    21   GpuCAReduce{add}{1}(GpuElemwise{Composite{[add(mul(scalar\_sigmoid(i0), i1, i2, i3), true\_div(mul(i4, i5), i6))]}}[(0, 0)].0)
253. 7.1%    46.9%       0.205s       2.05e-05s   10000    17   GpuAlloc(GpuDimShuffle{0}.0, Shape\_i{0}.0)
254. 6.9%    53.8%       0.198s       1.98e-05s   10000     4   GpuElemwise{sub,no\_inplace}(CudaNdarrayConstant{[ 1.]}, y)
255. 6.9%    60.7%       0.198s       1.98e-05s   10000    12   GpuElemwise{inv,no\_inplace}(GpuFromHost.0)
256. 6.2%    66.9%       0.178s       1.78e-05s   10000    11   GpuElemwise{neg,no\_inplace}(GpuElemwise{Composite{[sub(neg(i0), i1)]}}[(0, 0)].0)
257. 5.6%    72.5%       0.159s       1.59e-05s   10000    19   GpuElemwise{Composite{[add(mul(scalar\_sigmoid(i0), i1, i2, i3), true\_div(mul(i4, i5), i6))]}}[(0, 0)](GpuElemwise{Composite{[sub(neg(i0), i1)]}}[(0, 0)].0, CudaNdarrayConstant{[-1.]}, GpuAlloc.0, y, GpuElemwise{ScalarSigmoid}[(0, 0)].0, GpuElemwise{sub,no\_inplace}.0, GpuFromHost.0)
258. 4.8%    77.3%       0.138s       1.38e-05s   10000    18   HostFromGpu(GpuElemwise{ScalarSigmoid}[(0, 0)].0)
259. 4.4%    81.7%       0.126s       1.26e-05s   10000    10   GpuFromHost(Elemwise{Cast{float32}}.0)
260. 4.3%    86.0%       0.124s       1.24e-05s   10000     9   GpuElemwise{Composite{[sub(neg(i0), i1)]}}[(0, 0)](GpuGemv{inplace}.0, GpuDimShuffle{x}.0)
261. 4.2%    90.2%       0.121s       1.21e-05s   10000    15   GpuElemwise{ScalarSigmoid}[(0, 0)](GpuElemwise{neg,no\_inplace}.0)
262. 4.2%    94.4%       0.119s       1.19e-05s   10000    23   GpuElemwise{Composite{[sub(i0, mul(i1, i2))]}}[(0, 0)](b, CudaNdarrayConstant{0.00999999977648}, GpuCAReduce{add}{1}.0)
263. 3.4%    97.7%       0.096s       9.61e-06s   10000    16   HostFromGpu(GpuElemwise{Composite{[sub(mul(i0, scalar\_softplus(i1)), mul(i2, i3, scalar\_softplus(i4)))]},no\_inplace}.0)
264. 0.5%    98.2%       0.013s       1.33e-06s   10000    20   Elemwise{gt,no\_inplace}(HostFromGpu.0, TensorConstant{(1,) of 0.5})
265. 0.3%    98.5%       0.010s       9.81e-07s   10000     2   GpuDimShuffle{1,0}(x)
266. 0.3%    98.8%       0.008s       8.27e-07s   10000     1   Shape\_i{0}(x)
267. 0.3%    99.1%       0.008s       7.90e-07s   10000    14   GpuDimShuffle{0}(GpuElemwise{inv,no\_inplace}.0)
268. ... (remaining 16 Apply instances account for 0.90%(0.03s) of the runtime)

271. # 3. Conclusions
273. Examine and compare 'Ops' summaries for CPU and GPU. Usually GPU ops 'GpuFromHost' and 'HostFromGpu' by themselves
274. consume a large amount of extra time, but by making as few as possible data transfers between GPU and CPU, you can minimize their overhead.
275. Notice that each of the GPU ops consumes more time than its CPU counterpart. This is because the ops operate on small inputs;
276. if you increase the input data size (e.g. set N = 4000), you will see a gain from using the GPU.
278. """

# 二、 GpuArray Backend

    如果你还没有准备好，你需要安装 libgpuarray 和至少一个计算工具箱。可以看相关的介绍说明 [libgpuarray](http://deeplearning.net/software/libgpuarray/installation.html).

    如果使用OpenGL，那么所有设备的类型都支持的，对于该章节剩下的部分，不管你使用的计算设备是什么，都表示是gpu。

waring：我们想完全支持OpenCL, 在2014年5月的时候，该支持仍然是个想法而已。一些有用的ops仍然没有被支持，因为 想要在旧的后端以最小化变化来移植。

### 2.1 Testing Theano with GPU

    为了查看是否使用的是GPU，可以将下面代码剪切然后创建个文件运行：

1. **from** theano **import** function, config, shared, tensor, sandbox
2. **import** numpy
3. **import** time
5. vlen = 10 \* 30 \* 768  # 10 x #cores x # threads per core
6. iters = 1000
8. rng = numpy.random.RandomState(22)
9. x = shared(numpy.asarray(rng.rand(vlen), config.floatX))
10. f = function([], tensor.exp(x))
11. **print** f.maker.fgraph.toposort()
12. t0 = time.time()
13. **for** i **in** xrange(iters):
14. r = f()
15. t1 = time.time()
16. **print** 'Looping %d times took' % iters, t1 - t0, 'seconds'
17. **print** 'Result is', r
18. **if** numpy.any([isinstance(x.op, tensor.Elemwise) **and**
19. ('Gpu' **not** **in** type(x.op).\_\_name\_\_)
20. **for** x **in** f.maker.fgraph.toposort()]):
21. **print** 'Used the cpu'
22. **else**:
23. **print** 'Used the gpu'

    该程序只计算一群随机数的 exp() 。注意到我们使用 [**theano.shared()**](http://deeplearning.net/software/theano/library/index.html#theano.shared) 函数来确保输入x存储在gpu上。

1. $ THEANO\_FLAGS=device=cpu python check1.py
2. [Elemwise{exp,no\_inplace}(<TensorType(float64, vector)>)]
3. Looping 1000 times took 2.6071999073 seconds
4. Result **is** [ 1.23178032  1.61879341  1.52278065 ...,  2.20771815  2.29967753
5. 1.62323285]
6. Used the cpu
8. $ THEANO\_FLAGS=device=cuda0 python check1.py
9. Using device cuda0: GeForce GTX 275
10. [GpuElemwise{exp,no\_inplace}(<GpuArray<float64>>), HostFromGpu(gpuarray)(GpuElemwise{exp,no\_inplace}.0)]
11. Looping 1000 times took 2.28562092781 seconds
12. Result **is** [ 1.23178032  1.61879341  1.52278065 ...,  2.20771815  2.29967753
13. 1.62323285]
14. Used the gpu

### 2.2 返回在设备上分配数据的句柄

    在默认情况下，在gpu上执行的函数仍然返回一个标准的numpy ndarray。在得到结果之前会有一个迁移操作，将数据传输会cpu上从而来确保与cpu代码的兼容。这可以让在不改变源代码的情况下只使用flag device来改变代码运行的位置。

    如果不建议损失一些灵活性，可以让theano直接返回gpu对象。下面的代码就是这样：

1. **from** theano **import** function, config, shared, tensor, sandbox
2. **import** numpy
3. **import** time
5. vlen = 10 \* 30 \* 768  # 10 x #cores x # threads per core
6. iters = 1000
8. rng = numpy.random.RandomState(22)
9. x = shared(numpy.asarray(rng.rand(vlen), config.floatX))
10. f = function([], sandbox.gpuarray.basic\_ops.gpu\_from\_host(tensor.exp(x)))
11. **print** f.maker.fgraph.toposort()
12. t0 = time.time()
13. **for** i **in** xrange(iters):
14. r = f()
15. t1 = time.time()
16. **print** 'Looping %d times took' % iters, t1 - t0, 'seconds'
17. **print** 'Result is', numpy.asarray(r)
18. **if** numpy.any([isinstance(x.op, tensor.Elemwise) **and**
19. ('Gpu' **not** **in** type(x.op).\_\_name\_\_)
20. **for** x **in** f.maker.fgraph.toposort()]):
21. **print** 'Used the cpu'
22. **else**:
23. **print** 'Used the gpu'

    这里的 **theano.sandbox.gpuarray.basic.gpu\_from\_host()** 调用的意思是 “将输入复制到 GPU上”。然而在优化的阶段中，因为结果已经在gpu上了，它会被移除掉（即该函数会被忽略）。这里是为了告诉theano我们想要gpu上的结果。

输出为：

1. $ THEANO\_FLAGS=device=cuda0 python check2.py
2. Using device cuda0: GeForce GTX 275
3. [GpuElemwise{exp,no\_inplace}(<GpuArray<float64>>)]
4. Looping 1000 times took 0.455810785294 seconds
5. Result **is** [ 1.23178032  1.61879341  1.52278065 ...,  2.20771815  2.29967753
6. 1.62323285]
7. Used the gpu

    然而每次调用的时间看上去会比之前的两个调用更少 (的确是会更少，因为这里避免了数据传输r)这里这么大的加速是因为gpu上执行的异步过程所导致的，也就是说工作并没有完成，只是“启动”了。

    返回的对象是一个从pygou上得到的 GpuArray。它几乎扮演着带有一些异常的 numpy ndarray ，因为它的数据都在gpu上，你可以将它复制到主机上，然后通过使用平常的numpy cast ，例如numpy.asarray()来转换成一个常规的ndarray 。

为了更快的速度，可以使用borrow flag，查阅： [Borrowing when Constructing Function Objects](http://deeplearning.net/software/theano/tutorial/aliasing.html#borrowfunction).

### 2.3 什么能够在gpu上加速？

    当然在不同设备之间，性能特性还是不太的，同样的，我们会改进我们的实现。

    该backend支持所有的常规theano数据类型 (float32, float64, int, ...)，然而GPU的支持是变化的，而且一些单元没法处理 double (float64)或者更小的 (小于32 位，比如 int16)数据类型。如果使用了这些单元，那么会在编译的时候或者运行的时候得到一个错误。

    复杂的支持还未测试，而且大多数都不行。

    通常来说，大的操作，比如矩阵乘法或者有着大量输入的逐元素操作将会明显更快的。

### 2.4 GPU 异步功能

    默认情况下，在gpu上所有的操作都是异步的，这可以通过底层的libgpuarray来使得这些操作都是透明的。

    当在设备和主机之间进行内存迁移的时候，可以通过引入同步点。当在gpu上释放活动的（活动的缓冲区就是仍然会被kernel使用的缓冲区）内存缓冲区的时候，可以引入另一个同步点。

    可以通过调用它的sync（）方法来对一个特定的GpuArray强制同步。这在做基准的时候可以用来得到准确的耗时计算。

    强制的同步点会和中间结果的垃圾回收相关联。为了得到最快的速度，你应该通过使用theano flag allow\_gc=False来禁用垃圾回收器。不过要注意这会导致内存使用提升的问题。

# 三、直接对gpu编程的一些软件

撇开theano这种元编程，有：

* **CUDA**: GPU 编程API，是NVIDIA 对C的扩展 (CUDA C)
  + 特定供应商
  + 成熟的数值库 (BLAS, RNG, FFT) 。
* **OpenCL**: CUDA的多供应商版本
  + 更加的通用和标准。
  + 更少的库，传播不广
* **PyCUDA**:对CUDA驱动接口的python绑定，允许通过python来访问 Nvidia的 CUDA 并行计算API
  + 方便:

使用python来更容易的进行GPU 元编程。

从python中能够抽象的编译更低层的 CUDA 代码 (pycuda.driver.SourceModule).

GPU 内存缓存 (pycuda.gpuarray.GPUArray).

帮助文档.

* + 完整性: 绑定了所有的CUDA驱动 API.
  + 自动的错误检测：所有的 CUDA 错误都会自动的转到python异常。
  + 速度: PyCUDA的底层是用 C++写的。
  + 针对GPU对象，具有很好的内存管理：

对象的清理是和对象的生命周期绑定的 (RAII, ‘Resource Acquisition Is Initialization’).

使得更容易编写正确的，无漏洞的和不容易崩溃的代码。

PyCUDA 会知道依赖条件 (例如，它不会在所有分配的内存释放之前对上下文进行分离)。

* (查阅PyCUDA的 [documentation](http://documen.tician.de/pycuda/index.html) 和 在PyCUDA上Andreas Kloeckner的 [website](http://mathema.tician.de/software/pycuda) )
* **PyOpenCL**: PyCUDA for OpenCL

## 四、学习用PyCUDA编程

    如果你已经精通C了，那么你就可以很容易的通过学习来充分利用你的知识，首先用CUDA C来编写GPU，然后，使用 PyCUDA来访问 CUDA API。

下面的资源有助于你学习的过程：

* **CUDA API 和CUDA C: 入门**
  + [NVIDIA’s slides](http://www.sdsc.edu/us/training/assets/docs/NVIDIA-02-BasicsOfCUDA.pdf)
  + [Stein’s (NYU) slides](http://www.cs.nyu.edu/manycores/cuda_many_cores.pdf)
* **CUDA API 和 CUDA C: 高级**
  + [MIT IAP2009 CUDA](https://sites.google.com/site/cudaiap2009/home) (full coverage: lectures, leading Kirk-Hwu textbook, 例子，额外的资源)
  + [Course U. of Illinois](http://courses.engr.illinois.edu/ece498/al/index.html) (full lectures, Kirk-Hwu 教科书)
  + [NVIDIA’s knowledge base](http://www.nvidia.com/content/cuda/cuda-developer-resources.html) (覆盖范围广，从入门到高级)
  + [practical issues](http://stackoverflow.com/questions/2392250/understanding-cuda-grid-dimensions-block-dimensions-and-threads-organization-s) ( grids, blocks 和 threads之间的关系；并在同一页还有相对应的问题)
  + [CUDA optimisation](http://www.gris.informatik.tu-darmstadt.de/cuda-workshop/slides.html)
* **PyCUDA: 入门**
  + [Kloeckner’s slides](http://www.gputechconf.com/gtcnew/on-demand-gtc.php?sessionTopic=&searchByKeyword=kloeckner&submit=&select=+&sessionEvent=2&sessionYear=2010&sessionFormat=3)
  + [Kloeckner’ website](http://mathema.tician.de/software/pycuda)
* **PYCUDA: 高级**
  + [PyCUDA documentation website](http://documen.tician.de/pycuda/)

    下面的例子是用来说明用PyCUDA来对GPU编程的一个预言。一旦你觉得完全足够了，你就可以尝试去做相对应的练习。

**Example: PyCUDA**

1. # (from PyCUDA's documentation)
2. **import** pycuda.autoinit
3. **import** pycuda.driver as drv
4. **import** numpy
6. **from** pycuda.compiler **import** SourceModule
7. mod = SourceModule("""
8. \_\_global\_\_ void multiply\_them(float \*dest, float \*a, float \*b)
9. {
10. const int i = threadIdx.x;
11. dest[i] = a[i] \* b[i];
12. }
13. """)
15. multiply\_them = mod.get\_function("multiply\_them")
17. a = numpy.random.randn(400).astype(numpy.float32)
18. b = numpy.random.randn(400).astype(numpy.float32)
20. dest = numpy.zeros\_like(a)
21. multiply\_them(
22. drv.Out(dest), drv.In(a), drv.In(b),
23. block=(400,1,1), grid=(1,1))
25. **assert** numpy.allclose(dest, a\*b)
26. **print** dest

### Exercise

    运行之前的例子

    修改并执行一个shape（20，10）的矩阵

**Example: Theano + PyCUDA**

1. **import** numpy, theano
2. **import** theano.misc.pycuda\_init
3. **from** pycuda.compiler **import** SourceModule
4. **import** theano.sandbox.cuda as cuda
6. **class** PyCUDADoubleOp(theano.Op):
7. **def** \_\_eq\_\_(self, other):
8. **return** type(self) == type(other)
10. **def** \_\_hash\_\_(self):
11. **return** hash(type(self))
13. **def** \_\_str\_\_(self):
14. **return** self.\_\_class\_\_.\_\_name\_\_
16. **def** make\_node(self, inp):
17. inp = cuda.basic\_ops.gpu\_contiguous(
18. cuda.basic\_ops.as\_cuda\_ndarray\_variable(inp))
19. **assert** inp.dtype == "float32"
20. **return** theano.Apply(self, [inp], [inp.type()])
22. **def** make\_thunk(self, node, storage\_map, \_, \_2):
23. mod = SourceModule("""
24. \_\_global\_\_ void my\_fct(float \* i0, float \* o0, int size) {
25. int i = blockIdx.x\*blockDim.x + threadIdx.x;
26. if(i<size){
27. o0[i] = i0[i]\*2;
28. }
29. }""")
30. pycuda\_fct = mod.get\_function("my\_fct")
31. inputs = [storage\_map[v] **for** v **in** node.inputs]
32. outputs = [storage\_map[v] **for** v **in** node.outputs]
34. **def** thunk():
35. z = outputs[0]
36. **if** z[0] **is** None **or** z[0].shape != inputs[0][0].shape:
37. z[0] = cuda.CudaNdarray.zeros(inputs[0][0].shape)
38. grid = (int(numpy.ceil(inputs[0][0].size / 512.)), 1)
39. pycuda\_fct(inputs[0][0], z[0], numpy.intc(inputs[0][0].size),
40. block=(512, 1, 1), grid=grid)
41. **return** thunk

使用这个代码来测试：

1. >>> x = theano.tensor.fmatrix()
2. >>> f = theano.function([x], PyCUDADoubleOp()(x))
3. >>> xv = numpy.ones((4, 5), dtype="float32")
4. >>> **assert** numpy.allclose(f(xv), xv\*2)
5. >>> **print** numpy.asarray(f(xv))

### Exercise

    运行前面的例子

    修改并执行两个矩阵的乘法： x \* y.

    修改并执行返回两个输出： x + y 和 x - y.

    (注意到theano当前的逐元素优化只对涉及到单一输出的计算有用。所以，为了提供基本解决情况下的效率，需要在代码中显式的对这两个操作进行优化)。

   修改然后执行来支持跨越行为（stride） (即，避免受限于输入一定是C-连续的)。

## 五、注意

   查阅 [Other Implementations](http://deeplearning.net/software/theano/tutorial/examples.html#example-other-random) 来了解如何在gpu上处理随机数

参考资料：

[1]官网：http://deeplearning.net/software/theano/tutorial/using\_gpu.html

## [Theano2.1.13-基础知识之PyCUDA、CUDAMat、Gnumpy的兼容](http://www.cnblogs.com/shouhuxianjian/p/4590223.html)

来自：http://deeplearning.net/software/theano/tutorial/gpu\_data\_convert.html

PyCUDA/CUDAMat/Gnumpy compatibility

# 一、PyCUDA

     当前，PyCUDA 和Theano使用不同的对象来存储GPU数据。这两种实现支持的是不同的特征集。 Theano的实现是叫做CudaNdarray ，并且支持strides。 同时只支持float32 dtype。 PyCUDA的实现叫做 GPUArray 而且不支持strides。 然而，它可以处理所有的NumPy 和CUDA dtypes。

    我们现在来介绍下如何工作在这两个都有的基对象上，而且也在模仿NumPy。下面有一些资料关于如何在同一个脚本中使用这两个对象。

### 1.1 迁移

      你可以使用 theano.misc.pycuda\_utils 模块来对 GPUArray和CudaNdarray之间进行转换。函数 to\_cudandarray(x,copyif=False) 和 to\_gpuarray(x) 返回一个新的对象，该对象占据着和原始对象同一块内存空间。不过它会抛出一个值错误（ValueError）的异常。因为GPUArrays不支持strides，如果CudaNdarray 是strided，那么我们需要对它进行non-strided复制。生成的GPUArray不会在共享同一片内存区域。如果你想要这种行为，那么可以在to\_gpuarray中设置 copyif=True 。

### 1.2 用PyCUDA 来编译

      你可以使用 PyCUDA来编译直接工作在CudaNdarrays上的 CUDA 函数。这里是来自文件theano/misc/tests/test\_pycuda\_theano\_simple.py中的例子：

1. **import** sys
2. **import** numpy
3. **import** theano
4. **import** theano.sandbox.cuda as cuda\_ndarray
5. **import** theano.misc.pycuda\_init
6. **import** pycuda
7. **import** pycuda.driver as drv
8. **import** pycuda.gpuarray

11. **def** test\_pycuda\_theano():
12. """Simple example with pycuda function and Theano CudaNdarray object."""
13. **from** pycuda.compiler **import** SourceModule
14. mod = SourceModule("""
15. \_\_global\_\_ void multiply\_them(float \*dest, float \*a, float \*b)
16. {
17. const int i = threadIdx.x;
18. dest[i] = a[i] \* b[i];
19. }
20. """)
22. multiply\_them = mod.get\_function("multiply\_them")
24. a = numpy.random.randn(100).astype(numpy.float32)
25. b = numpy.random.randn(100).astype(numpy.float32)
27. # Test with Theano object
28. ga = cuda\_ndarray.CudaNdarray(a)
29. gb = cuda\_ndarray.CudaNdarray(b)
30. dest = cuda\_ndarray.CudaNdarray.zeros(a.shape)
31. multiply\_them(dest, ga, gb,
32. block=(400, 1, 1), grid=(1, 1))
33. **assert** (numpy.asarray(dest) == a \* b).all()

### 1.3 Theano 操作，使用一个PyCUDA函数

    你可以在theano op中使用 用PyCUDA编译好的GPU函数：

1. **import** numpy, theano
2. **import** theano.misc.pycuda\_init
3. **from** pycuda.compiler **import** SourceModule
4. **import** theano.sandbox.cuda as cuda
6. **class** PyCUDADoubleOp(theano.Op):
7. **def** \_\_eq\_\_(self, other):
8. **return** type(self) == type(other)
9. **def** \_\_hash\_\_(self):
10. **return** hash(type(self))
11. **def** \_\_str\_\_(self):
12. **return** self.\_\_class\_\_.\_\_name\_\_
13. **def** make\_node(self, inp):
14. inp = cuda.basic\_ops.gpu\_contiguous(
15. cuda.basic\_ops.as\_cuda\_ndarray\_variable(inp))
16. **assert** inp.dtype == "float32"
17. **return** theano.Apply(self, [inp], [inp.type()])
18. **def** make\_thunk(self, node, storage\_map, \_, \_2):
19. mod = SourceModule("""
20. \_\_global\_\_ void my\_fct(float \* i0, float \* o0, int size) {
21. int i = blockIdx.x \* blockDim.x + threadIdx.x;
22. if(i<size){
23. o0[i] = i0[i] \* 2;
24. }
25. }""")
26. pycuda\_fct = mod.get\_function("my\_fct")
27. inputs = [ storage\_map[v] **for** v **in** node.inputs]
28. outputs = [ storage\_map[v] **for** v **in** node.outputs]
29. **def** thunk():
30. z = outputs[0]
31. **if** z[0] **is** None **or** z[0].shape!=inputs[0][0].shape:
32. z[0] = cuda.CudaNdarray.zeros(inputs[0][0].shape)
33. grid = (int(numpy.ceil(inputs[0][0].size / 512.)),1)
34. pycuda\_fct(inputs[0][0], z[0], numpy.intc(inputs[0][0].size),
35. block=(512, 1, 1), grid=grid)
36. thunk.lazy = False
37. **return** thunk

# 二、CUDAMat

        这里的函数是用来在CUDAMat对象和 Theano的 CudaNdArray对象之间进行转换的。 它们遵循和theano的PyCUDA函数一样的原则，可以查阅 theano.misc.cudamat\_utils.py.

    WARNING: 在这些转换器上，会有一个与stride/shape相关的特殊的问题。为了能够work，需要 transpose和reshape.等操作..

# 三、Gnumpy

   这是介于Gnumpy garray 对象和 Theano CudaNdArray 对象之间的转换函数。也同样相似于 Theano的 PyCUDA 函数，可查阅： theano.misc.gnumpy\_utils.py.

参考资料：

[1] 官网：http://deeplearning.net/software/theano/tutorial/gpu\_data\_convert.html

## [Theano2.1.14-基础知识之理解为了速度和正确性的内存别名](http://www.cnblogs.com/shouhuxianjian/p/4590222.html)

来自：http://deeplearning.net/software/theano/tutorial/aliasing.html

Understanding Memory Aliasing for Speed and Correctness

    内存的重用是theano提升代码运行速度的一种方法，而且理解theano如何别名（alias）缓冲区对编写的程序速度的提升和正确性的保证很重要。

    这部分是基于theano处理内存的基础上来说明原则的，而且解释了为了获得更快的性能，在什么时候该改变某些函数默认的行为和方法。

# 一、内存模型:两个空间

     可以通过一些简单的原则来指导theano的函数处理。首先，theano会管理一个内存池，并且在这个池中theano会追踪值的变化。

* Theano 会管理它自己的内存空间，并且通常不会与非theano代码的python变量的内存相重叠。
* Theano 函数只修改在theano内存空间中的缓冲区。
* Theano的内存空间包括分配给存储shared变量和临时用来执行函数的缓冲区。
* 物理意义上来说，theano的内存空间会跨越主机，GPU设备，而且在未来可能会涉及到远程机器上的对象。
* 分配给shared变量的内存缓冲区是唯一的：它不会被另一个shared变量所别名（aliased）。
* Theano管理的内存在theano函数未运行和theano的库代码未运行的时候是保持不变的。
* 一个函数的默认行为是返回用户空间的变量作为输出，然后期待用户空间的变量作为输入。

    介于theano管理的内存和用户管理的内存的区别可以通过一些theano函数（例如，shared，get\_value 和针对于In和Out的构造函数）和 borrow=True flag来细分。这可以让一些方法在冒着微妙的bug的风险下在整体的程序 (通过别名内存)上更快 (避免了复制操作) 。

    该章节剩下的部分意在帮助你理解什么时候使用 borrow=True 参数是安全的，并且可以让代码更快的运行。

# 二、Borrowing when 创建共享变量

     borrow 可以作为共享变量构造函数的参数：

1. **import** numpy, theano
2. np\_array = numpy.ones(2, dtype='float32')
4. s\_default = theano.shared(np\_array)
5. s\_false   = theano.shared(np\_array, borrow=False)
6. s\_true    = theano.shared(np\_array, borrow=True)

    默认情况(s\_default) 和显式的设置borrow=False 这两种情况下，构造的共享变量是对np\_array进行深度复制的。所以我们后续对np\_array的改变不会影响到这两个共享变量。

1. np\_array += 1 # now it is an array of 2.0 s改变的操作
3. s\_default.get\_value()  # -> array([1.0, 1.0])
4. s\_false.get\_value()    # -> array([1.0, 1.0])
5. s\_true.get\_value()     # -> array([2.0, 2.0])

    如果我们在cpu上运行这段代码，那么我们对np\_array的改变可以通过 s\_true.get\_value()看到，因为Numpy arrays是可变的，而且s\_ture使用了np\_array对象作为它的内部缓冲。

    然而，对np\_array的别名和 s\_true 没法保证这种情况一定发生，也许会临时性的发生，也许根本不发生。 没法保证是因为当theano使用的是gpu的时候，那么borrow 这个flag是没有任何影响的。只能临时性的发生是因为如果我们调用一个theano函数来更新s\_ture的值，那么这个别名关系也许会，也许不会被打破 (该函数允许通过修改它的缓冲区来更新这个shared变量，这会保留这个别名，或者改变这个变量指向的缓冲区，而这会终止这个别名）。

**要点（Take home message:）：**

   当shared变量代表一个大的对象的时候，在shared变量的构造函数中使用borrow=True是安全的操作（也是好主意） (在内存占用方面) ，而且你不需要在内存中对它进行复制。

    想要利用副作用，从而通过使用 borrow=True 来修改 shared 变量的方法不是一个别名技术，因为在一些设备上 (例如GPU 设备)该技术不会生效的。

# 三、Borrowing when 访问共享变量的值

检索

     borrow 参数可以同样用来控制如何检索 shared 变量的值。

1. s = theano.shared(np\_array)
3. v\_false = s.get\_value(borrow=False) # N.B. borrow default is False
4. v\_true = s.get\_value(borrow=True)

    当 borrow=False 传递给 get\_value， 的时候，意味着返回的值不会是对theano的内部的内存进行别名。当borrow=True 传递给 get\_value的时候，意味着返回的值可能是对theano的内部的内存的别名。不过这两种调用都会对内部的内存进行复制，产生副本。

   使用 borrow=True 还仍然会进行复制的原因是因为shared变量的内部的表达并不是和你想的一样。当你通过传递一个NumPy 数组来创建一个shared变量的时候，例如，然后 get\_value() 也必须返回一个NumPy 数组。这就是为什么theano可以让gpu的使用看起来透明的原因。不过当你使用gpu的时候(或在未来可能是一个远程机器)，那么 numpy.ndarray 就不是你数据的中间表示了。如果你真的想要theano返回它的中间表示且不想对它进行复制，那么你应该对get\_value函数使用return\_internal\_type=True 参数。它将不会 cast内部对象 (总是在常量时间内返回的)，不过也许会由环境因素（例如：计算设备，Numpy数组的dtype）而返回各种不同的数据类型。

1. v\_internal = s.get\_value(borrow=True, return\_internal\_type=True)

   可以将 borrow=False 和 return\_internal\_type=True结合起来使用 ，这会返回内部对象的一个深度复制。这对内部的调试来说是很重要的，而不是对通常的使用而言的。

    我们可以透明使用theano对不同类型的优化，有个原则就是当shared变量创建之后， get\_value() 在默认情况下总是会返回和它接受的同一个对象类型。所以如果你在gpu上手动创建了数据，然后用这个数据在gpu上创建一个共享变量。 当return\_internal\_type=False的时候get\_value总是会返回gpu上的数据。

**要点（Take home message:）：**

      当你的代码没有修改返回的值的时候使用get\_value(borrow=True) 是安全的 (而且有时候也更快)。不要使用副作用（side-effect）来修改一个“shared”变量，因为它会让你的代码具有设备依赖。通过副作用的方法来修改gpu变量是不可能的。

### 分配

    Shared 变量同样有一个 set\_value 方法可以接受一个可选的 borrow=True 参数。该语义相似于那些创建新shared变量的语义， borrow=False 是默认的，而且 borrow=True 意思是theano可能会重用作为变量内部存储的缓冲区。

    手动更新shared变量的值的一个标准的模式是：

1. s.set\_value(
2. some\_inplace\_fn(s.get\_value(borrow=True)),
3. borrow=True)

      该模式工作的时候是不管计算设备是什么的，当后者（应该说的是设备）可能会在没有复制的情况下暴露theano的内部变量，那么它就会和in-place一样快的速度进行更新。

    当在gpu上分配 shared 变量的时候，gpu与主机之间的内存迁移是很耗时的。这里是一些提示，用于却跑如何快速和高效的使用gpu的内存和带宽：

* 在Theano 0.3.1之前， set\_value 在gpu上不是以in-place方式工作的。也就是说，有时候gpu对于旧的内存没有释放之前就对新变量进行内存的分配了。如果你在gpu内存上运行的已经接近于上限了，这可能会导致你得到gpu内存已经耗尽的提示。

解决方法：更新到最新的theano。

* 如果你打算反复的对一个共享变量的进出数据块进行交换，你可能会想要重用第一次分配的内存I，这可以更快而且更好的使用内存

         解决方法： 更新到最新版本的theano (>0.3.0)并考虑 padding源数据来确保每个块都有着相同的size。

* 同样值得提到的就是，当前gpu的复制只支持连续的内存。所以theano必须先将你提供的值转换成c-连续形式，然后在对它进行复制。这需要额外的在主机上对数据进行复制。

解决方法：确保你想要赋值给 CudaNdarraySharedVariable 的数据已经是C-连续了。

    可以在[sandbox.cuda.var – The Variables for Cuda-allocated arrays](http://deeplearning.net/software/theano/library/sandbox/cuda/var.html#libdoc-cuda-var)上面找到当前gpu版本的set\_value()的实现过程。

# 四、Borrowing when 构造函数对象

     borrow 参数同样可以提供给 In 和 Out 对象来控制theano.function 是如何处理它的参数 argument[s] 和返回值value[s]的：

1. **import** theano, theano.tensor
3. x = theano.tensor.matrix()
4. y = 2 \* x
5. f = theano.function([theano.In(x, borrow=True)], theano.Out(y, borrow=True))

    Borrowing 一个输入意味着theano会将你提供的参数暂时的视为是theano池的一部分。因此，在执行函数（例如，f）的时候，在对其他变量进行计算的过程中，你的输入可能会作为一个缓冲区而重用（和重写）。

    Borrowing 一个输出意味着theano不会在每次调用函数的时候坚持分配一个新的输出缓冲区。它可能会在之前的调用的基础上重用同一个，并重写旧的内容。因而，它会通过副作用来重写旧的返回值。这些返回值也会在执行另一个编译好的函数上被重写 (例如，输出会被别名成一个shared变量)。所以在调用更多的theano函数之前，小心使用一个 borrowed 返回值。默认情况下当然是不borrow 内部结果的。

    同样可以传递一个 return\_internal\_type=True flag 给 Out 变量，该变量有着和对shared变量的get\_value函数设置return\_internal\_type flag时一样的解释。不同于 get\_value(), return\_internal\_type=True 和 borrow=True 参数的结合，然后传给 Out() 不能保证说避开了对一个输出值的复制。他们只是隐式的对graph的编译和优化提供了更多的灵活性。

对于 GPU的graphs来说，borrowing是影响速度的主要因素：

1. **from** theano **import** function, config, shared, sandbox, tensor, Out
2. **import** numpy
3. **import** time
5. vlen = 10 \* 30 \* 768  # 10 x # cores x # threads per core
6. iters = 1000
8. rng = numpy.random.RandomState(22)
9. x = shared(numpy.asarray(rng.rand(vlen), config.floatX))
10. f1 = function([], sandbox.cuda.basic\_ops.gpu\_from\_host(tensor.exp(x)))
11. f2 = function([],
12. Out(sandbox.cuda.basic\_ops.gpu\_from\_host(tensor.exp(x)),
13. borrow=True))
14. t0 = time.time()
15. **for** i **in** xrange(iters):
16. r = f1()
17. t1 = time.time()
18. no\_borrow = t1 - t0
19. t0 = time.time()
20. **for** i **in** xrange(iters):
21. r = f2()
22. t1 = time.time()
23. **print** 'Looping', iters, 'times took', no\_borrow, 'seconds without borrow',
24. **print** 'and', t1 - t0, 'seconds with borrow.'
25. **if** numpy.any([isinstance(x.op, tensor.Elemwise) **and**
26. ('Gpu' **not** **in** type(x.op).\_\_name\_\_)
27. **for** x **in** f1.maker.fgraph.toposort()]):
28. **print** 'Used the cpu'
29. **else**:
30. **print** 'Used the gpu'

结果：

1. $ THEANO\_FLAGS=device=gpu0,floatX=float32 python test1.py
2. Using gpu device 0: GeForce GTX 275
3. Looping 1000 times took 0.368273973465 seconds without borrow **and** 0.0240728855133 seconds with borrow.
4. Used the gpu

**要点（Take home message:）：**

    当在函数返回值后，输入x 对于函数来说是不需要的，你可以将它做为额外的工作空间，然后考虑使用 In(x, borrow=True)来对它进行标记 。它可以让函数变得更快，而且减少对内存的需求。当一个返回值y 很大（内存占用方面)，你只需要当它返回的时候，读取它一次然后考虑对它进行标记 Out(y, borrow=True)。

参考资料：

[1] 官网：http://deeplearning.net/software/theano/tutorial/aliasing.html

## [Theano2.1.15-基础知识之theano如何处理shapre信息](http://www.cnblogs.com/shouhuxianjian/p/4590221.html)

来自：http://deeplearning.net/software/theano/tutorial/shape\_info.html

How Shape Information is Handled by Theano

   当建立一个graph的时候，是没法严格的执行theano变量的shape的，因为在运行的时候将具体的值作为提供给一个theano函数的参数来说，也许会受到它graph中theano变量的shape的限制。

当前，有两种方式来使用shape信息:

* 当准确的输出shape已经提前知道的情况下，为了在cpu和gpu上让2d卷积得到更快的c代码。
* 当我们只想要知道shape，而不是一个变量的实际的值的时候，也就是为了移除graph中的计算。可以使用方法： [Op.infer\_shape](http://deeplearning.net/software/theano/extending/cop.html#Op.infer_shape) 。

例子：

1. >>> **import** theano
2. >>> x = theano.tensor.matrix('x')
3. >>> f = theano.function([x], (x \*\* 2).shape)
4. >>> theano.printing.debugprint(f)
5. MakeVector [@A] ''   2
6. |Shape\_i{0} [@B] ''   1
7. | |x [@C]
8. |Shape\_i{1} [@D] ''   0
9. |x [@C]

    这个编译后的函数的输出不包含任何乘法或幂次运算。theano已经将它们移除，直接计算输出的shape。

# 一、Shape的推测问题

    Theano 在graph中传播关于shape的信息。有时候，这会导致错误，考虑下面的例子：

1. >>> **import** numpy
2. >>> **import** theano
3. >>> x = theano.tensor.matrix('x')
4. >>> y = theano.tensor.matrix('y')
5. >>> z = theano.tensor.join(0, x, y)
6. >>> xv = numpy.random.rand(5, 4)
7. >>> yv = numpy.random.rand(3, 3)
8. >>> f = theano.function([x,y], z.shape)
9. >>> theano.printing.debugprint(f)
10. MakeVector [@A] ''   4
11. |Elemwise{Add}[(0, 0)] [@B] ''   3
12. | |Shape\_i{0} [@C] ''   1
13. | | |x [@D]
14. | |Shape\_i{0} [@E] ''   2
15. |   |y [@F]
16. |Shape\_i{1} [@G] ''   0
17. |x [@D]

    print f(xv,yv)# 不应该引起错误。[8, 4]

1. >>> f = theano.function([x,y], z)# Do not take the shape.
2. >>> theano.printing.debugprint(f)
3. Join [@A] ''   0
4. |TensorConstant{0} [@B]
5. |x [@C]
6. |y [@D]
7. >>> f(xv,yv)
8. >>> # Raises a dimensions mismatch error.

    正如你看到的，当只想要知道一些计算的shape的时候 (例子中的join )，可以在没有执行该计算本身(在第一个输出和调试打印的时候没有 join )的时候，直接得到一个推测的shape 。

    这使得shape的计算更快，不过它同样也隐藏着错误。在这个例子中，join的输出的shape的计算只基于第一个输入的theano变量得到的，这会引发一个错误。

    当使用其他ops的时候，例如 elemwise 和 dot，错误也许会发生。确实，为了执行一些优化（速度或者稳定性），theano首先假设计算是正确的。

    你可以通过运行没有优化的代码来检测这些问题，使用theano flagoptimizer\_excluding=local\_shape\_to\_shape\_i。你同样可以通过在模式FAST\_COMPILE(它不会使用这个优化，也不会使用其他的大部分优化) 或者 DebugMode (它会在所有优化之后来进行测试（更慢）)下运行来得到同样的结果。

# 二、指定准确的shape

    当前，指定一个shape梅雨哦和我们想的那样容易和灵活。我们打算进行一些升级。这里就是当前我们可以做到的：

* 你可以在调用conv2d函数的时候，直接将shape信息传递给 ConvOp 。简单的在调用的时候设置参数为  image\_shape 和filter\_shape 。他们必须是4个元素的元组。例如：

1. theano.tensor.nnet.conv2d(..., image\_shape=(7, 3, 5, 5), filter\_shape=(2, 3, 4, 4))

* 你可以使用 SpecifyShape 操作来在graph的任何位置上增加shape信息。这可以允许执行某些优化。在下面的例子中，这使得对theano函数进行预计算成为一个常量。

1. >>> **import** theano
2. >>> x = theano.tensor.matrix()
3. >>> x\_specify\_shape = theano.tensor.specify\_shape(x, (2, 2))
4. >>> f = theano.function([x], (x\_specify\_shape \*\* 2).shape)
5. >>> theano.printing.debugprint(f)
6. DeepCopyOp [@A] ''   0
7. |TensorConstant{(2,) of 2} [@B]

# 三、未来的计划

    参数 “constant shape”可以加入到 theano.shared()中 。这可能是在使用shared变量的时候最频繁的操作了。它会让代码更简单，而且可可以当更新shared变量的时候使得shape不发生改变。

参考资料：

[1]官网：http://deeplearning.net/software/theano/tutorial/shape\_info.html

## [Theano2.1.16-基础知识之调试：常见的问题解答](http://www.cnblogs.com/shouhuxianjian/p/4590220.html)

来自：http://deeplearning.net/software/theano/tutorial/shape\_info.html

Debugging Theano: FAQ and Troubleshooting

    在计算机程序中会有许多种不同的bug。该页就是来说说FAQ，即问题集的。介绍了一些处理常见问题的方法，并介绍了一些在我们自己的theano代码中，用于查找问题（即使该问题发生在theano内部）的工具： [Using DebugMode](http://deeplearning.net/software/theano/tutorial/modes.html#using-debugmode).

# 一、将问题独立出来/测试theano的编译器

    你可以在 [*DebugMode*](http://deeplearning.net/software/theano/tutorial/modes.html#using-debugmode) 下运行thenao的函数。该模式下会测试theano的优化，并有助于找到问题的所在，例如NaN，inf 和其他问题。

# 二、分析错误信息

    甚至在默认的配置下，theano都会尝试显示有用的错误信息。考虑下面的错误代码：

1. **import** numpy as np
2. **import** theano
3. **import** theano.tensor as T
5. x = T.vector()
6. y = T.vector()
7. z = x + x
8. z = z + y
9. f = theano.function([x, y], z)
10. f(np.ones((2,)), np.ones((3,)))

运行上面的代码：

1. Traceback (most recent call last):
2. File "test0.py", line 10, **in** <module>
3. f(np.ones((2,)), np.ones((3,)))
4. File "/PATH\_TO\_THEANO/theano/compile/function\_module.py", line 605, **in** \_\_call\_\_
5. self.fn.thunks[self.fn.position\_of\_error])
6. File "/PATH\_TO\_THEANO/theano/compile/function\_module.py", line 595, **in** \_\_call\_\_
7. outputs = self.fn()
8. ValueError: Input dimension mis-match. (input[0].shape[0] = 3, input[1].shape[0] = 2)
9. Apply node that caused the error: Elemwise{add,no\_inplace}(<TensorType(float64, vector)>, <TensorType(float64, vector)>, <TensorType(float64, vector)>)
10. Inputs types: [TensorType(float64, vector), TensorType(float64, vector), TensorType(float64, vector)]
11. Inputs shapes: [(3,), (2,), (2,)]
12. Inputs strides: [(8,), (8,), (8,)]
13. Inputs scalar values: ['not scalar', 'not scalar', 'not scalar']
15. 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 optimization can be disabled with 'optimizer=None'.
16. HINT: Use the Theano flag 'exception\_verbosity=high' **for** a debugprint of this apply node.

    可以说最有用的信息通常差不多一半都来自对错误信息的分析理解，而且错误信息也是按照引起错误的顺序显示的 (ValueError: 输入维度不匹配. (input[0].shape[0] = 3, input[1].shape[0] = 2).。在它下面，给出了一些其他的信息，例如apply节点导致的错误，还有输入类型，shapes，strides 和scalar values。

    最后两个提示在调试的时候也是很有用的。使用theano flag optimizer=fast\_compile 或者 optimizer=None 可以告诉你出错的那一行，而 exception\_verbosity=high 会显示apply节点的调试打印（debugprint）。使用这些提示，错误信息最后会变成：

1. Backtrace when the node **is** created:
2. File "test0.py", line 8, **in** <module>
3. z = z + y
5. Debugprint of the apply node:
6. Elemwise{add,no\_inplace} [@A] <TensorType(float64, vector)> ''
7. |Elemwise{add,no\_inplace} [@B] <TensorType(float64, vector)> ''
8. | |<TensorType(float64, vector)> [@C] <TensorType(float64, vector)>
9. | |<TensorType(float64, vector)> [@C] <TensorType(float64, vector)>
10. |<TensorType(float64, vector)> [@D] <TensorType(float64, vector)>

    这里我们可以看到错误可以追溯到 z = z + y这一行 。对于这个例子来说，使用 optimizer=fast\_compile 是有效果的，如果它没效果，你就需要设置 optimizer=None 或者使用测试值。

# 三、使用测试值

   在 v.0.4.0版本的时候，Theano有一个新机制，也就是theano.function 编译之前，graph是动态执行的。因为优化在这个阶段还没执行，所以对于用户来说就很容易定位bug的来源。这个功能可以通过配置flagtheano.config.compute\_test\_value启用。下面这个例子就很好的说明了这点。这里，我们使用exception\_verbosity=high 和 optimizer=fast\_compile，这里（个人：该例子中）不会告诉你具体出错的那一行（个人在；这里与上面有些矛盾，不过看得出来这里提示出错的是调用的函数，而上面出错定位到了语句。具体的留待以后在分析）。 optimizer=None 因而就很自然的用来代替测试值了。

1. **import** numpy
2. **import** theano
3. **import** theano.tensor as T
5. # compute\_test\_value is 'off' by default, meaning this feature is inactive
6. theano.config.compute\_test\_value = 'off' # Use 'warn' to activate this feature
8. # configure shared variables
9. W1val = numpy.random.rand(2, 10, 10).astype(theano.config.floatX)
10. W1 = theano.shared(W1val, 'W1')
11. W2val = numpy.random.rand(15, 20).astype(theano.config.floatX)
12. W2 = theano.shared(W2val, 'W2')
14. # input which will be of shape (5,10)
15. x  = T.matrix('x')
16. # provide Theano with a default test-value
17. #x.tag.test\_value = numpy.random.rand(5, 10)
19. # transform the shared variable in some way. Theano does not
20. # know off hand that the matrix func\_of\_W1 has shape (20, 10)
21. func\_of\_W1 = W1.dimshuffle(2, 0, 1).flatten(2).T
23. # source of error: dot product of 5x10 with 20x10
24. h1 = T.dot(x, func\_of\_W1)
26. # do more stuff
27. h2 = T.dot(h1, W2.T)
29. # compile and call the actual function
30. f = theano.function([x], h2)
31. f(numpy.random.rand(5, 10))

    运行上面的代码，生成下面的错误信息：

1. Traceback (most recent call last):
2. File "test1.py", line 31, **in** <module>
3. f(numpy.random.rand(5, 10))
4. File "PATH\_TO\_THEANO/theano/compile/function\_module.py", line 605, **in** \_\_call\_\_
5. self.fn.thunks[self.fn.position\_of\_error])
6. File "PATH\_TO\_THEANO/theano/compile/function\_module.py", line 595, **in** \_\_call\_\_
7. outputs = self.fn()
8. ValueError: Shape mismatch: x has 10 cols (**and** 5 rows) but y has 20 rows (**and** 10 cols)
9. Apply node that caused the error: Dot22(x, DimShuffle{1,0}.0)
10. Inputs types: [TensorType(float64, matrix), TensorType(float64, matrix)]
11. Inputs shapes: [(5, 10), (20, 10)]
12. Inputs strides: [(80, 8), (8, 160)]
13. Inputs scalar values: ['not scalar', 'not scalar']
15. Debugprint of the apply node:
16. Dot22 [@A] <TensorType(float64, matrix)> ''
17. |x [@B] <TensorType(float64, matrix)>
18. |DimShuffle{1,0} [@C] <TensorType(float64, matrix)> ''
19. |Flatten{2} [@D] <TensorType(float64, matrix)> ''
20. |DimShuffle{2,0,1} [@E] <TensorType(float64, 3D)> ''
21. |W1 [@F] <TensorType(float64, 3D)>
23. 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 optimization can be disabled with 'optimizer=None'.

    如果上面的信息还不够， 可以通过改变一些代码，从而让theano来揭示错误的准确来源。

1. # enable on-the-fly graph computations
2. theano.config.compute\_test\_value = 'warn'
4. ...
6. # input which will be of shape (5, 10)
7. x  = T.matrix('x')
8. # provide Theano with a default test-value
9. x.tag.test\_value = numpy.random.rand(5, 10)

    上面的代码中，我们将符号矩阵x 赋值一个特定的测试值。这允许theano按照之前定义的那样，动态的执行符号表达式（通过对每个op调用perform方法）。因此，可以在编译通道中更准确和更早的识别到错误的来源。例如，运行上面的代码得到下面的错误信息，正确的识别到了第24行。

1. Traceback (most recent call last):
2. File "test2.py", line 24, **in** <module>
3. h1 = T.dot(x, func\_of\_W1)
4. File "PATH\_TO\_THEANO/theano/tensor/basic.py", line 4734, **in** dot
5. **return** \_dot(a, b)
6. File "PATH\_TO\_THEANO/theano/gof/op.py", line 545, **in** \_\_call\_\_
7. required = thunk()
8. File "PATH\_TO\_THEANO/theano/gof/op.py", line 752, **in** rval
9. r = p(n, [x[0] **for** x **in** i], o)
10. File "PATH\_TO\_THEANO/theano/tensor/basic.py", line 4554, **in** perform
11. z[0] = numpy.asarray(numpy.dot(x, y))
12. ValueError: matrices are **not** aligned

 compute\_test\_value 机制如下方式工作：

* 当使用Theano的 constants 和 shared 变量的时候，不需要instrument它们。
* 一个theano变量 (例如： dmatrix, vector,等等) 应该通过属性 tag.test\_value来赋值特定的测试值。
* Theano 会自动instruments 中间的结果。所以，任何从x中得到的值会自动由tag.test\_value引用。

compute\_test\_value 可以有以下的值:

* off: 默认行为. 这时候调试机制是未激活的。
* raise:动态计算测试值。任何变量都需要一个测试值，不过不需要用户来提供，这被认为是一个错误。会相应的抛出一个异常。
* warn: Idem, 发出一个警告，而不是抛出异常。
* ignore: 当一个变量没有测试值的时候，会静默的忽略掉中间测试值的计算。

note：该特性暂时不能与 Scan 兼容，而且也无法和那些没有实现perform方法的ops相兼容。

# 四、我如何在一个函数中输出中间值？

    Theano提供了一个‘Print’ 操作：

1. x = theano.tensor.dvector('x')
3. x\_printed = theano.printing.Print('this is a very important value')(x)
5. f = theano.function([x], x \* 5)
6. f\_with\_print = theano.function([x], x\_printed \* 5)
8. #this runs the graph without any printing
9. **assert** numpy.all( f([1, 2, 3]) == [5, 10, 15])
11. #this runs the graph with the message, and value printed
12. **assert** numpy.all( f\_with\_print([1, 2, 3]) == [5, 10, 15])

    因为 Theano 是以拓扑顺序来运行你的程序的，你没法准确的按照顺序来控制，这时候多个Print()是同时运行的。想要知道更详细的关于在哪里、什么时候、怎样计算的，查阅： [“How do I Step through a Compiled Function?”](http://deeplearning.net/software/theano/tutorial/debug_faq.html#faq-monitormode).

warning：使用这个Print Theano 操作可以防止一些theano的优化。这也可以在稳定的优化的时候使用，所以如果你使用这个Print，然后有NaN，那么就试着移除它们来看看是否是它们导致的错误 。

# 五、我如何在编译前后输出一个graph

    Theano 提供两个函数 (**theano.pp()** 和 [**theano.printing.debugprint()**](http://deeplearning.net/software/theano/library/printing.html#theano.printing.debugprint)) 来在编译的前后打印graph到终端上。这两个函数以不同的方式来打印表达式： **pp()** 更紧凑，而且更像数学； **debugprint()** 更详细 。Theano 同样提供 [**theano.printing.pydotprint()**](http://deeplearning.net/software/theano/library/printing.html#theano.printing.pydotprint) ，这会生成一副关于函数的png图片。

更详细的查阅： [printing – Graph Printing and Symbolic Print Statement](http://deeplearning.net/software/theano/library/printing.html#libdoc-printing).

# 六、我编译的函数太慢了，怎么办？

    首先，确保你运行在 FAST\_RUN 模式下。虽然 FAST\_RUN 是默认情况下的模式，不过还是坚持要传递 mode='FAST\_RUN' 给theano.function (或者 theano.make) 或者设置[**config.mode**](http://deeplearning.net/software/theano/library/config.html#config.mode) 为 FAST\_RUN.

其次，尝试 Theano [ProfileMode](http://deeplearning.net/software/theano/tutorial/modes.html#using-profilemode). 这会告诉你现在是哪个 Apply 节点和哪个ops在你的cpu周期上。

提示:

* 使用flags floatX=float32 来请求类型 float32 而不是 float64; 使用 Theano 构造函数matrix(),vector(),... 而不是 dmatrix(), dvector(),... 因为他们分别涉及到默认的类型 float32 和 float64.
* 当你想要以相同的类型来将两个矩阵进行相乘的时候，记得以profile模式来检查在编译后的graph中没有Dot操作。当输入是矩阵而且有着相同的类型的时候，Dot会被优化成dot22。当然在使用floatX=float32 ，而且其中一个graph的输入是类型float64的时候也是这样。

# 七、我如何对一个编译后的函数进行step调试

    你可以使用 MonitorMode 来检查当函数被调用的时候每个节点的输入和输出。下面的代码就展示了如何打印所有的输入和输出：

1. **import** theano
3. **def** inspect\_inputs(i, node, fn):
4. **print** i, node, "input(s) value(s):", [input[0] **for** input **in** fn.inputs],
6. **def** inspect\_outputs(i, node, fn):
7. **print** "output(s) value(s):", [output[0] **for** output **in** fn.outputs]
9. x = theano.tensor.dscalar('x')
10. f = theano.function([x], [5 \* x],
11. mode=theano.compile.MonitorMode(
12. pre\_func=inspect\_inputs,
13. post\_func=inspect\_outputs))
14. f(3)
16. # The code will print the following:
17. #   0 Elemwise{mul,no\_inplace}(TensorConstant{5.0}, x) input(s) value(s): [array(5.0), array(3.0)] output(s) value(s): [array(15.0)]

    当在 MonitorMode的情况下，使用 inspect\_inputs 和 inspect\_outputs 这些函数。你应该看到 [可能很多] 打印的输出。每个 Apply 节点都会被打印出来，按照graph中的位置顺序，参数到函数 perform 或者 c\_code 和计算得到的输出。不可否认，如果你使用的是大张量，这会有着超多的输出要读... 不过你可以选择增加逻辑来打印一部分信息，比如打印那些用到某种op的，在程序的某个位置，或者在输入或者输出上的一个具体的值。一个典型的例子就是检测什么时候NaN的值会被加到计算中，如下面代码：

1. **import** numpy
3. **import** theano
5. # This is the current suggested detect\_nan implementation to
6. # show you how it work.  That way, you can modify it for your
7. # need.  If you want exactly this method, you can use
8. # ``theano.compile.monitormode.detect\_nan`` that will always
9. # contain the current suggested version.
11. **def** detect\_nan(i, node, fn):
12. **for** output **in** fn.outputs:
13. **if** (**not** isinstance(output[0], numpy.random.RandomState) **and**
14. numpy.isnan(output[0]).any()):
15. **print** '\*\*\* NaN detected \*\*\*'
16. theano.printing.debugprint(node)
17. **print** 'Inputs : %s' % [input[0] **for** input **in** fn.inputs]
18. **print** 'Outputs: %s' % [output[0] **for** output **in** fn.outputs]
19. **break**
21. x = theano.tensor.dscalar('x')
22. f = theano.function([x], [theano.tensor.log(x) \* x],
23. mode=theano.compile.MonitorMode(
24. post\_func=detect\_nan))
25. f(0)  # log(0) \* 0 = -inf \* 0 = NaN
27. # The code above will print:
28. #   \*\*\* NaN detected \*\*\*
29. #   Elemwise{Composite{[mul(log(i0), i0)]}} [@A] ''
30. #    |x [@B]
31. #   Inputs : [array(0.0)]
32. #   Outputs: [array(nan)]

    为了帮助理解在你的graph中在发生的的事情，你可以禁用 local\_elemwise\_fusion 和所有的 inplace 优化。首先是速度优化，也就是会将逐元素操作融合到一起的优化。这会使的更难知道哪个具体的逐元素导致的问题。第二个优化就是会让某些ops的输出重写它们的输入。所以如果一个op生成一个坏的输出，你就没法看到在post\_func函数中被重写之前的输入。为了禁用这些优化（0.6rc3之后的版本），如下定义MonitorMode：

1. mode = theano.compile.MonitorMode(post\_func=detect\_nan).excluding(
2. 'local\_elemwise\_fusion', 'inplace)
3. f = theano.function([x], [theano.tensor.log(x) \* x],
4. mode=mode)

note：Theano flags optimizer\_including, optimizer\_excluding 和 optimizer\_requiring 不会被 MonitorMode使用的，它们只会在default模式下使用。当你想要定义监视的部分的时候，你没法将 default 模式和MonitorMode一起使用。

    为了确保所有的节点的输入都是在调用到psto\_func的时候可用的，你必须同样禁用垃圾回收。执行的节点垃圾回收那些theano函数不再需要的输入。这可以通过下面的flag来指定：

1. allow\_gc=False

# 八、我如何使用pdb

   在大部分情况下，你不是在交互模式下执行程序而是以python脚本的方式。在这种情况下，对python调试器的使用就变得十分的需要了，特别是当你的模型变得更加复杂的时候。中间的结果不需要有很清晰的名字，而且你会得到那些很那解读的异常，因为这是函数编译后的自然特性导致的：

考虑这个例子脚本 (“ex.py”):

1. **import** theano
2. **import** numpy
3. **import** theano.tensor as T
5. a = T.dmatrix('a')
6. b = T.dmatrix('b')
8. f = theano.function([a, b], [a \* b])
10. # matrices chosen so dimensions are unsuitable for multiplication
11. mat1 = numpy.arange(12).reshape((3, 4))
12. mat2 = numpy.arange(25).reshape((5, 5))
14. f(mat1, mat2)

    这实际上如此的简单，而且调试也是如此的容易，不过这是为了图文讲解的目的。 正如矩阵没法逐元素相乘（不匹配的shapes），我们得到了下面的异常：

1. File "ex.py", line 14, **in** <module>
2. f(mat1, mat2)
3. File "/u/username/Theano/theano/compile/function\_module.py", line 451, **in** \_\_call\_\_
4. File "/u/username/Theano/theano/gof/link.py", line 271, **in** streamline\_default\_f
5. File "/u/username/Theano/theano/gof/link.py", line 267, **in** streamline\_default\_f
6. File "/u/username/Theano/theano/gof/cc.py", line 1049, **in** execute ValueError: ('Input dimension mis-match. (input[0].shape[0] = 3, input[1].shape[0] = 5)', Elemwise{mul,no\_inplace}(a, b), Elemwise{mul,no\_inplace}(a, b))

    调用的堆栈包含着一些有用的信息，从而可以追溯错误的来源。 首先是编译后的函数被调用的脚本– 不过如果你使用（不正确的参数化）预建立模块，错误也许来自这些模块中的ops，而不是这个脚本。最后一行告诉我们这个op引起了这个异常。一个“mul”涉及到变量“a”和“b”。不过这里假设我们替换了一个没有名字的中间值。

    在了解了theano中graph结构的一些知识，我们可以使用python调试器来探索这个graph，然后我们就可以得到运行时的错误信息。特别是矩阵维度对指出错误的来源很有用。在打印出的结果中，会涉及到矩阵的4维度中的2个维度，不过因为例子的原因，我们需要其他的维度来指出错误。首先我们再次运行调试器模块，然后用“c”来运行该程序：

1. python -m pdb ex.py
2. > /u/username/experiments/doctmp1/ex.py(1)<module>()
3. -> **import** theano
4. (Pdb) c

    然后我们返回到上面错误的打印输出部分，不过解释器停留在了那个状态。有用的命令如下：

* “up” 和 “down” (往下或往上移动这个调用堆栈),
* “l” (在当前堆栈位置上打印该行周围的代码),
* “p variable\_name” (打印 ‘variable\_name’的字符串解释),
* “p dir(object\_name)”, 使用python的 dir() 函数来打印一个对象的成员的列表。

    例如，键入 “up”,和一个简单的 “l” 会得到一个局部变量 “node”。 该 “node” 来自于计算graph中，所以通过跟随 “node.inputs”, “node.owner” 和 “node.outputs” 连接，就能探索这个graph。

    这个graph是纯符号的 (没有数据，只有抽象的符号操作)。为了得到实际参数的信息，你需要探索 “thunk” 对象，这是通过函数自身（一个“thunk”就是一个关于闭包的概念）来绑定了输入（和输出）的存储的 。这里，为了得到当前节点的第一个输入shape，你需要键入 “p thunk.inputs[0][0].shape”，这会打印出 “(3, 4)”.

# 九、Dumping 一个函数来帮助调试

    如果你读到这里了，那么就可能是你邮件到了我们的主列表，然后我们建议你读的这部分。这部分解释了如何dump所有的传到theano.function()的参数。这有助于帮助我们在编译的时候复制问题，然这并不要求你举一个自圆其说的例子。

    为了让这工作起来，我们需要导入graph中所有op的代码，所以如果你创建了你自己的op，我们需要这份代码。然而，我们不会unpickle它，我们已经有了来自theano和Pylearn2的所有Ops：

1. # Replace this line:
2. theano.function(...)
3. # with
4. theano.function\_dump(filename, ...)
5. # Where filename is a string to a file that we will write to.

然后和我们说文件名。

参考资料：

[1]官网：http://deeplearning.net/software/theano/tutorial/shape\_info.html

[**Theano2.1.17-基础知识之剖析theano的函数**](http://www.cnblogs.com/shouhuxianjian/p/4590219.html)

来自：http://deeplearning.net/software/theano/tutorial/profiling.html

Profiling Theano function

note：该方法是用来代替旧的ProfileMode.不要再使用ProfileMode了。

         在检查错误的同时，另一个重要的任务就是剖析你的代码。你会用到theano 的flags 或者参数，然后将它们传递给 [**theano.function**](http://deeplearning.net/software/theano/library/compile/function.html#function.function)**。**

        最简单的剖析theano函数的方式就是使用下面介绍的theano flags。当进程存在的时候，它们会将信息输出到stdout（标准输出流）。

使用 ProfileMode 是一个三步过程

开启这个分析器是很简单的，只需要用到 flag [**config.profile**](http://deeplearning.net/software/theano/library/config.html#config.profile).

确保内存分析器用到theano 的flag: [**config.profile\_memory**](http://deeplearning.net/software/theano/library/config.html#config.profile_memory) 和 [**config.profile**](http://deeplearning.net/software/theano/library/config.html#config.profile).

为了能够在theano优化的时候进行分析，使用theano 的flag: [**config.profile\_optimizer**](http://deeplearning.net/software/theano/library/config.html#config.profile_optimizer) 和 [**config.profile**](http://deeplearning.net/software/theano/library/config.html#config.profile).

你可以使用theano 的flags **profiling.n\_apply**、**profiling.n\_ops** 和**profiling.min\_memory\_size** 来修改打印信息的数量。

分析器会对每个theano函数进行分析，而且会分析打印出的分析的总和。每个分析包含4个部分：全局信息，类信息，ops信息和apply节点信息。

    在全局部分， “Message” 就是theano函数的名称， theano.function() 有一个可选的参数 name 而这个默认是为None。对name进行有意义的赋值有助于你分析许多theano函数 。在这个部分中，我们同样看到函数调用的次数和在所有这些调用上花费的总的时间。花费在Function.fn.\_\_call\_\_ 上和在块中的时间分析有助于理解theano的开销。

    同样的，我们看到在编译过程的两个阶段上时间的花费：优化(修改graph使得能够更加的稳定/快速) 和链接(编译c 代码并可以被python调用).

    类，ops和apply节点部分有着相同的信息：关于运行的apply节点的信息。ops部分会从apply部分得到有用的信息然后融合那些有着相同op的apply节点。如果在graph中两个apply节点有着两个相同的ops，那么它们会被融合。一些节点比如逐元素，如果它们的参数不同的话（被执行的标量），就不相等了。 所以类部分会比ops部分融合更多的apply节点。

    这里就是一个当我们禁用某些theano优化来更加直观清晰的理解不同的部分之间的差异的一个例子。在当所有的优化都启用的时候，那么在graph中就只剩下一个op了。

note：

为了剖析在gpu上内存使用的高峰情况，你需要：

\* 在文件中 theano/sandbox/cuda/cuda\_ndarray.cu, 设置宏

COMPUTE\_GPU\_MEM\_USED to 1.

\* 然后调用 theano.sandbox.cuda.theano\_allocated()

它返回一个有着2个int值的元组.第一个值就是指示当前的gpu分配给theano的内存；第二个就是theano占用gpu内存使用的峰值。

不要总是开启这个宏，这会减慢内存分配和释放。而且还会减慢计算速度，所以会影响到速度分析。所以不要在这些情况下开启这个。

运行这个例子:

THEANO\_FLAGS=optimizer\_excluding=fusion:inplace,profile=True python doc/tutorial/profiling\_example.py

输出：

1. Function profiling
2. ==================
3. Message: None
4. Time **in** 1 calls to Function.\_\_call\_\_: 5.698204e-05s
5. Time **in** Function.fn.\_\_call\_\_: 1.192093e-05s (20.921%)
6. Time **in** thunks: 6.198883e-06s (10.879%)
7. Total compile time: 3.642474e+00s
8. Theano Optimizer time: 7.326508e-02s
9. Theano validate time: 3.712177e-04s
10. Theano Linker time (includes C, CUDA code generation/compiling): 9.584920e-01s
12. Class
13. ---
14. <% time> <sum %> <apply time> <time per call> <type> <#call> <#apply> <Class name>
15. 100.0%   100.0%       0.000s       2.07e-06s     C        3        3   <**class** 'theano.tensor.elemwise.Elemwise'>
16. ... (remaining 0 Classes account **for**   0.00%(0.00s) of the runtime)
18. Ops
19. ---
20. <% time> <sum %> <apply time> <time per call> <type> <#call> <#apply> <Op name>
21. 65.4%    65.4%       0.000s       2.03e-06s     C        2        2   Elemwise{add,no\_inplace}
22. 34.6%   100.0%       0.000s       2.15e-06s     C        1        1   Elemwise{mul,no\_inplace}
23. ... (remaining 0 Ops account **for**   0.00%(0.00s) of the runtime)
25. Apply
26. ------
27. <% time> <sum %> <apply time> <time per call> <#call> <id> <Apply name>
28. 50.0%    50.0%       0.000s       3.10e-06s      1     0   Elemwise{add,no\_inplace}(x, y)
29. 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)
30. 15.4%   100.0%       0.000s       9.54e-07s      1     1   Elemwise{add,no\_inplace}(Elemwise{add,no\_inplace}.0, z)
31. ... (remaining 0 Apply instances account **for** 0.00%(0.00s) of the runtime)

参考资料：

[1]官网：http://deeplearning.net/software/theano/tutorial/profiling.html

## [Theano2.1.18-基础知识之theano的扩展](http://www.cnblogs.com/shouhuxianjian/p/4594532.html)

来自：http://deeplearning.net/software/theano/tutorial/extending\_theano.html

Extending Theano

    该教程覆盖了如何使用新颖的ops来扩展theano。它主要关注哪些能够提供一个python实现的ops。而[Extending Theano with a C Op](http://deeplearning.net/software/theano/tutorial/extending_theano_c.html#extending-theano-c) 是基于c的op实现。该教程的第一部分介绍了theano的graphs，因为提供一个新颖的theano op需要对theano graphs有个基本的理解。然后在概括的介绍了定义一个op的最重要的方法。

    正如一个图文说明需要的，该教程会介绍如何编写一个简单的基于python的op，该op是基于double上执行的。同样会介绍如何实现测试来确保之前op的工作的有效性。

note：

该教程不会介绍如何编写得到输入值的视图或者对其修改的op。所以这里介绍的所有ops必须返回新的分配的内存或者重用由函数[perform()](http://deeplearning.net/software/theano/extending/op.html#perform) 的参数output\_storage 所提供的内存。 想了解这些过程可以查阅 [Views and inplace operations](http://deeplearning.net/software/theano/extending/inplace.html#views-and-inplace) 。

如果你的op返回针对输入的视图或者修改，却不是如介绍的那样实现的，那么theano就还是会运行，并且返回某些graphs的正确的值和其他graphs的错误结果。

强烈建议在DebugMode (Theano flag mode=DebugMode) 下运行你的测试，因为它会验证你的op行为是否正确。

note：查阅 [Developer Start Guide](http://deeplearning.net/software/theano/dev_start_guide.html#dev-start-guide) 来了解版本框架的信息，即git 和GitHub,和开发工作流程以及如何进行有质量的贡献。

### 一、Theano Graphs

### [../_images/apply_node.png](http://deeplearning.net/software/theano/_images/apply_node.png)

    Theano会将符号数学化计算表示成graph。这些graph都是二分图 （有2种类型node的图），它们由内连接的[Apply](http://deeplearning.net/software/theano/extending/graphstructures.html#apply) 和 [Variable](http://deeplearning.net/software/theano/extending/graphstructures.html#variable) 节点组成。 [Variable](http://deeplearning.net/software/theano/extending/graphstructures.html#variable) 节点在graph中表示数据，例如：输入、输出或者中间值。所以一个graph的输入和输出也就是theano [Variable](http://deeplearning.net/software/theano/extending/graphstructures.html#variable) 节点的列表. [Apply](http://deeplearning.net/software/theano/extending/graphstructures.html#apply) 节点通过在这些变量上执行计算来生成新的值。每个 [Apply](http://deeplearning.net/software/theano/extending/graphstructures.html#apply) 节点会连接到 [Op](http://deeplearning.net/software/theano/extending/graphstructures.html#op) 的一个实例，从而来表示所执行的计算。该教程详细介绍了如何编写这样一个op实例。关于图结构的更详细信息，查阅： [Graph Structures](http://deeplearning.net/software/theano/extending/graphstructures.html#graphstructures) [翻译](http://blog.csdn.net/shouhuxianjian/article/details/46509271)。

### 二、Op 结构

       op就是继承自gof.Op的某个python对象。该部分提供了一些方法的概述，可以用来实现一个新的op。。不过没有介绍你可能遇到或者需要的所有可能的情况。不过更详细的可以查阅： [Op’s contract](http://deeplearning.net/software/theano/extending/op.html#op-contract).

[复制代码](javascript:void(0);)

import theano

class MyOp(theano.Op):

# Properties attribute

\_\_props\_\_ = ()

def make\_node(self, \*inputs):

pass

# Python implementation:

def perform(self, node, inputs\_storage, output\_storage):

pass

# Other type of implementation

# C implementation: [see theano web site for other functions]

def c\_code(...):

# ...

pass

# Other implementations (pycuda, ...):

def make\_thunk(self, node, storage\_map, \_, \_2):

pass

# optional:

check\_input = True

def \_\_init\_\_(self, ...):

pass

def grad(self, inputs, g):

pass

def R\_op(self, inputs, eval\_points):

pass

def infer\_shape(node, (i0\_shapes, ...)):

pass

[复制代码](javascript:void(0);)

op需要执行在gof.Op接口中定义的一些方法。对于op来说，首先需要定义方法[make\_node()](http://deeplearning.net/software/theano/extending/op.html#make_node)  然后在实现该方法，例如[perform()](http://deeplearning.net/software/theano/extending/op.html#perform), [Op.c\_code()](http://deeplearning.net/software/theano/extending/cop.html#Op.c_code) 或 [make\_thunk()](http://deeplearning.net/software/theano/extending/op.html#make_thunk)。

[make\_node()](http://deeplearning.net/software/theano/extending/op.html#make_node) 方法生成一个Apply节点来表示在提供的输入上op的应用。该方法需要对下面三个事情负责：

* 首先检查输入变量类型是否与当前的op兼容。如果在当前输入类型上没法应用，那么必须抛出一个异常 (例如 TypeError).
* 在theano的符号语言中，它会对\*inputs上找到的变量进行操作，从而推断出符号输出变量的类型。并创建一个适合的符号类型的输出变量来作为该操作的输出。
* 它创建一个有着输入和输出变量的apply实例，然后返回该apply实例。

[perform()](http://deeplearning.net/software/theano/extending/op.html#perform) 方法用来定义一个op的python实现。它有以下几个参数：

* node 是对一个Apply节点的引用，从前面的 Op‘s [make\_node()](http://deeplearning.net/software/theano/extending/op.html#make_node) 方法获得。它通常不会用在简单的ops中，不过它包含的符号信息需要复杂的ops。
* inputs 是对数据的引用列表，会被非符号语句所操作（即，python，Numpy中的语句）。
* output\_storage 是存储单元的列表，其中存储的是输出数据。每一个单元存储着op的一个输出。放入 output\_storage 的数据必须匹配符号输出的类型。禁止改变output\_storage中的列表的长度。函数模式可以允许output\_storage 中的元素在执行过程中保持不变，或者将output\_storage单元的值重设为None。而且允许对op所需要用到的一些内存进行预分配。该特性允许 perform 重用调用期间的内存，例如，假设在output\_storage中预先分配了一些内存，其中有着正确的dtype，不过shape却是错的，而且存在某些stride模式。

[perform()](http://deeplearning.net/software/theano/extending/op.html#perform) 方法必须由输入里决定。也就是说，当用在同一个输入的时候，该方法必须返回相同的输出。

gof.Op 允许使用其他的方法来定义op的实现。例如可以通过定义 [Op.c\_code()](http://deeplearning.net/software/theano/extending/cop.html#Op.c_code) 来提供一个对该op的c实现。查阅 [Extending Theano with a C Op](http://deeplearning.net/software/theano/tutorial/extending_theano_c.html#extending-theano-c) 来获得关于 [Op.c\_code()](http://deeplearning.net/software/theano/extending/cop.html#Op.c_code) 和其他相关的c方法更详细的介绍。注意到一个op可以提供c和pythong两种实现。

[make\_thunk()](http://deeplearning.net/software/theano/extending/op.html#make_thunk) 方法是另一个可以用来代替 [perform()](http://deeplearning.net/software/theano/extending/op.html#perform)的方法。它返回一个thunk.。该thunk被定义成一个0参数函数，该函数很好的封装了op在对应节点的参数上的计算过程。该方法有以下几个参数：

* node 是Apply的实例，其中会请求一个thunk。
* storage\_map 是列表的字典，通过将变量映射成一个元素列表（one-element lists）来处理变量的当前的值。这个一元素列表可以作为指向值的指针，可以与其他节点和实例来共享该”指针“。
* compute\_map 同样也是列表的字典。将变量映射成一元素列表来处理布尔值。如果该值是0，那么这个变量就表示还未计算，该值是无效的。如果该值是1，那么这个变量已经被计算过了，那么是有效的。如果该值是2，那么这个变量是已经被垃圾回收了，不再有效，而且无法被这次调用所请求。返回的函数必须确保计算值在compute\_map中计算过。

[make\_thunk()](http://deeplearning.net/software/theano/extending/op.html#make_thunk) 在你自己想要生成代码和编译的时候是很有用的。例如，这可以让你使用PyCUDA来编译GPU代码。

如果 [make\_thunk()](http://deeplearning.net/software/theano/extending/op.html#make_thunk) 被一个op所定义，那么它会被theano所使用来获得该op的实现。 [perform()](http://deeplearning.net/software/theano/extending/op.html#perform) 和 [Op.c\_code()](http://deeplearning.net/software/theano/extending/cop.html#Op.c_code)会被忽略掉。

其他通过op定义的可选的方法：

[\_\_str\_\_()](http://deeplearning.net/software/theano/extending/op.html#__str__) 方法对你的op提供了一个有意义的字符串表示。

[\_\_eq\_\_()](http://deeplearning.net/software/theano/extending/op.html#__eq__) 和 [\_\_hash\_\_()](http://deeplearning.net/software/theano/extending/op.html#__hash__) 分别定义了两个ops是否相等和计算一个op实例的哈希。它们会在优化阶段被用来融合节点，那些有着相同计算的节点（相同的输入，相同的操作）。两个ops可以通过 [\_\_eq\_\_()](http://deeplearning.net/software/theano/extending/op.html#__eq__) 来计算是否相等，即在有着相同输入的时候是否有着相同的输出。

[\_\_props\_\_](http://deeplearning.net/software/theano/extending/op.html#__props__) 列出的属性影响着计算是如何执行的 (通常来说，它们就是你在**\_\_init\_\_()**中设置的那样). 它必须是一个元组。如果你没有任何属性，那么你应该将这个属性设置成空元组（ tuple ()）。

[\_\_props\_\_](http://deeplearning.net/software/theano/extending/op.html#__props__) 能够自动的生成合适的 [\_\_eq\_\_()](http://deeplearning.net/software/theano/extending/op.html#__eq__) 和 [\_\_hash\_\_()](http://deeplearning.net/software/theano/extending/op.html#__hash__)。 给定从[\_\_props\_\_](http://deeplearning.net/software/theano/extending/op.html#__props__)生成的方法 [\_\_eq\_\_()](http://deeplearning.net/software/theano/extending/op.html#__eq__), 两个ops如果它们对于[\_\_props\_\_](http://deeplearning.net/software/theano/extending/op.html#__props__)中列出的所有属性都有有着相同的值，那么这两个ops就是相等的。给定从[\_\_props\_\_](http://deeplearning.net/software/theano/extending/op.html#__props__) 生成的方法 [\_\_hash\_\_()](http://deeplearning.net/software/theano/extending/op.html#__hash__) ,两个ops如果对于 [\_\_props\_\_](http://deeplearning.net/software/theano/extending/op.html#__props__)中列出的所有属性都有着相同的值，那么这两个ops有着相同的hash。 [\_\_props\_\_](http://deeplearning.net/software/theano/extending/op.html#__props__) 同样会对你的op生成一个合适的 [\_\_str\_\_()](http://deeplearning.net/software/theano/extending/op.html#__str__) 。这需要theano的版本为 September 1st, 2014 or version 0.7.

[infer\_shape()](http://deeplearning.net/software/theano/extending/op.html#infer_shape) 方法允许在没有实际计算输出的情况下，推断op输出变量的shape。它可以作为输入 node（一个对Apply节点的引用)和theano符号变量的列表 (i0\_shape, i1\_shape, ...)(op输入变量的shape)。 [infer\_shape()](http://deeplearning.net/software/theano/extending/op.html#infer_shape) 返回一个列表，其中每个元素都是一个元组，用来标识一个输出的shape。这在当你只需要输出的shape而无需计算实际的输出的时候是很有用的，例如在优化的时候。

[grad()](http://deeplearning.net/software/theano/extending/op.html#grad) 方法适用于你想要对一些包含你op的表达式的损失函数进行微分的时候。在这个方法中梯度可以具体的符号化。它有两个参数inputs 和 output\_gradients ，这两个都是符号的theano变量的列表，必须被theano符号化语言所操作。该grad方法必须返回一个列表，其中对应每个输入都有一个变量。每个返回的变量表示关于输入的梯度，而该梯度是基于关于每个输出的符号化梯度上计算得到的（个人：这句话估计理解有误）。如果输出不是关于一个输入的微分，那么该方法就会被定义成针对这个输入返回一个NullType类型的变量。同样的，如果你没有对某些输入实现grad计算，你需要返回关于这个输入的一个NullType类型的变量。查阅 [grad()](http://deeplearning.net/software/theano/extending/op.html#grad)来得到更详细的说明。

[R\_op()](http://deeplearning.net/software/theano/extending/op.html#R_op) 方法当你想要让 theano.tensor.Rop 和你的op一起工作的时候是需要使用的。该函数实现了op表示的函数的R-操作。假设函数为 f ，输入为 x 。使用R-操作，即计算f 的jacobian然后通过v 与其右乘，表达式即： \frac{\partial f}{\partial x} v.

可选的布尔 check\_input 属性用来指定你是否想要在你的op中使用这个类型来在c\_code中检查它们的输入。它可以用来加速计算，减少开销（特别是在标量上）和减少生成的c文件的数量。

### 三、Op 例子

1. [复制代码](javascript:void(0);)
2. import theano
4. class DoubleOp(theano.Op):
5. \_\_props\_\_ = ()
7. def make\_node(self, x):
8. # check that the theano version has support for \_\_props\_\_
9. assert hasattr(self, '\_props')
10. x = theano.tensor.as\_tensor\_variable(x)
11. return theano.Apply(self, [x], [x.type()])
13. def perform(self, node, inputs, output\_storage):
14. x = inputs[0]
15. z = output\_storage[0]
16. z[0] = x \* 2
18. def infer\_shape(self, node, i0\_shapes):
19. return i0\_shapes
21. def grad(self, inputs, output\_grads):
22. return [output\_grads[0] \* 2]
24. def R\_op(self, inputs, eval\_points):
25. # R\_op can receive None as eval\_points.
26. # That mean there is no diferientiable path through that input
27. # If this imply that you cannot compute some outputs,
28. # return None for those.
29. if eval\_points[0] is None:
30. return eval\_points

return self.grad(inputs, eval\_points)

[复制代码](javascript:void(0);)

可以按照如下形式来测试：

1. [复制代码](javascript:void(0);)
2. x = theano.tensor.matrix()
3. f = theano.function([x], DoubleOp()(x))
4. import numpy
5. inp = numpy.random.rand(5, 4)
6. out = f(inp)
7. assert numpy.allclose(inp \* 2, out)
8. print inp

print out

[复制代码](javascript:void(0);)

### 四、如何进行测试

      Theano有很多函数来简化测试。这些可以用来测试 infer\_shape, grad 和 R\_op 方法。将下面的代码放入一个文件中，然后用theano-nose

程序来执行该文件。

### 4.1 基本的测试

     基本的测试就是通过使用op，然后检查返回的是否正确。如果你检测到一个错误，那么你就需要抛出一个异常。你可以使用关键字assert来自动的抛出一个AssertionError。

1. [复制代码](javascript:void(0);)
2. from theano.tests import unittest\_tools as utt
3. from theano import config
4. class test\_Double(utt.InferShapeTester):
5. def setUp(self):
6. super(test\_Double, self).setUp()
7. self.op\_class = DoubleOp
8. self.op = DoubleOp()
10. def test\_basic(self):
11. x = theano.tensor.matrix()
12. f = theano.function([x], self.op(x))
13. inp = numpy.asarray(numpy.random.rand(5, 4), dtype=config.floatX)
14. out = f(inp)
15. # Compare the result computed to the expected value.

utt.assert\_allclose(inp \* 2, out)

[复制代码](javascript:void(0);)

    我们调用 utt.assert\_allclose(expected\_value, value) 来对比NumPy ndarray。这抛出一个有着更多信息的错误消息。同样的，这个默认的容忍可以通过theano flags config.tensor.cmp\_sloppy 来改变，其值为0, 1 和 2. 默认的值会有着最严格的比较，而1 和2 就比较不那么严格了。

### 4.2 测试infer\_shape

     当一个类继承自 InferShapeTester 类，那么就可以通过 self.\_compile\_and\_check 来测试op的 infer\_shape方法。在当只需要输出的shape而无需计算实际的输出的时侯，用来测试该op在graph中的优化情况。另外，它通过优化后的graph计算得到的shape与实际计算的输出的shape进行对比来检查当前的shape是否正确。      self.\_compile\_and\_check 会编译一个theano函数。它会将输入和输出的theano变量的列表作为参数（正如theano.function那样），然后将真实值传递给编译后的函数。 它同样会将op类作为一个参数，来保证在优化后的shape graph中没有实例出现。

    如果有错误，那么该函数抛出一个异常。如果你想要看错误的时候的行为，可以使用一个非正常的 infer\_shape.

    在测试的时候，如果在不同的维度上shape的值相同 (例如，一个方阵或者有着shape为 (n, n, n),或(m, n, m)的tensor3 )，那么就没法检测。例如，如果 infer\_shape 使用矩阵的 width来作为它的height，那么在方阵的情况下就没法检测是否出错了。这就是为什么 self.\_compile\_and\_check 在这种情况下会输出一个警告的原因。如果你的操作只在这种矩阵下工作，那么就可以通过 warn=False 参数来禁用该警告。

1. [复制代码](javascript:void(0);)
2. from theano.tests import unittest\_tools as utt
3. from theano import config
4. class test\_Double(utt.InferShapeTester):
5. # [...] as previous tests.
6. def test\_infer\_shape(self):
7. x = theano.tensor.matrix()
8. self.\_compile\_and\_check([x], # theano.function inputs
9. [self.op(x)], # theano.function outputs
10. # Always use not square matrix!
11. # inputs data
12. [numpy.asarray(numpy.random.rand(5, 4),
13. dtype=config.floatX)],
14. # Op that should be removed from the graph.

self.op\_class)

[复制代码](javascript:void(0);)

### 4.3 测试梯度

    函数 [verify\_grad](http://deeplearning.net/software/theano/extending/unittest.html#validating-grad) 用来验证一个op或者theano graph的梯度。它通过对比分析梯度 (通过符号计算得到的)梯度和数值梯度 (通过有限的微分方法得到的)。

    如果出错了，那么该函数抛出一个异常。如果你想要看出错的情况，你可以执行一个错误的梯度 (例如，移除前面的2)。

1. def test\_grad(self):
2. theano.tests.unittest\_tools.verify\_grad(self.op,
3. [numpy.random.rand(5, 7, 2)])

### 4.4 测试Rop

     类 RopLop\_checker 定义了函数**RopLop\_checker.check\_mat\_rop\_lop()**，RopLop\_checker.check\_rop\_lop() 和RopLop\_checker.check\_nondiff\_rop(). 这些函数可以用来测试一个具体的op的Rop方法的实现。

例如，为了验证DoubleOp的Rop方法，你可以如下来测试：

1. [复制代码](javascript:void(0);)
2. import numpy
3. import theano.tests
4. from theano.tests.test\_rop import RopLop\_checker
5. class test\_DoubleRop(RopLop\_checker):
6. def setUp(self):
7. super(test\_DoubleRop, self).setUp()
8. def test\_double\_rop(self):
9. self.check\_rop\_lop(DoubleRop()(self.x), self.in\_shape)

[复制代码](javascript:void(0);)

### 4.5 测试 GPU Ops

    在GPU上执行的Ops应该继承自 theano.sandbox.cuda.GpuOp 而不是 theano.Op。这可以给theano区分它们的机会。目前，我们使用这个在GPU上的和约间代码来测试NVIDIA驱动是否正确工作。

### 五、运行你的测试

   为了执行你的测试程序，你需要选择下面的某种方法 ：

### 5.1 theano-nose

    选择要进行测试的方法就是运行 theano-nose. 在常规的theano安装中，后面的参数会读取操作系统的path，然后直接访问其中的文件夹。而且它也可以在Theano/bin文件夹中被访问。对应于不同的目的，可以使用下面不同的命令：

* theano-nose --theano: 运行任何在theano的path上找到的测试。
* theano-nose folder\_name: 运行任何在文件夹folder\_name内找到的测试。
* theano-nose test\_file.py: 运行任何在文件test\_file.py内的测试。

下面的命令对于开发者来说特别有用，因为它们可以针对具体的类或者具体的测试被调用

* theano-nose test\_file.py:test\_DoubleRop: 运行在类 test\_DoubleRop中的测试。
* theano-nose test\_file.py:test\_DoubleRop.test\_double\_op: 只运行在类test\_DoubleRop中的测试test\_double\_op 。

针对于 theano-nose 的用法和功能的详细介绍，可以使用带有参数为 --help (-h)的命令。（个人：在windows下运行theano-nose会显示找不到该命令，在ubuntu下没问题，即该教程的所有命令其实都是基于linux的，而非windows的。）

**5.2 nosetests**

    命令 nosetests 也是很有用的。虽然它缺少和 theano-nose 提供的信息， nosetests 的用法和theano-nose 相似，可以测试任何在python的路径上的文件夹：

nosetests [suffix similar to the above].

更多有关 nosetests 的资料，可查阅： [nosetests](http://readthedocs.org/docs/nose/en/latest/).

**5.3 In-file**

   可以增加一块代码在文件的最后，然后运行该文件。在这个例子中，在类test\_double\_op中的测试 test\_DoubleRop 会运行的。

1. if \_\_name\_\_ == '\_\_main\_\_':
2. t = test\_DoubleRop("test\_double\_rop")
3. t.setUp()
4. t.test\_double\_rop()

我们建议如果在运行一个文件的时候，最好运行那个文件中所有的测试，这可以通过在测试文件的最后加上如下的代码：

1. if \_\_name\_\_ == '\_\_main\_\_':
2. unittest.main()

### 六、练习

    运行上面的 DoubleOp e例子的代码。

   修改并执行： x \* y.

    修改并执行： x + y and x - y.

    你可以省略Rop函数。 试着实现上面的测试过程。

(注意到当前的theano的逐元素风格的优化只能用在涉及到单一输出的情况下。所以，为了解决这个问题，需要在代码中对这两个操作进行联合的显式优化。)

### 七、as\_op

    as\_op是一个pythong修饰符，用来将一个python函数转换成一个基础theano op，该op会在执行的时候调用所提供的函数。

    不推荐用该方法来建立一个op，不过这却是一个快速实现的方法。

    它有一个可选的 [infer\_shape()](http://deeplearning.net/software/theano/extending/op.html#infer_shape) 参数，该参数必须有如下的形式：

1. def infer\_shape(node, input\_shapes):
2. # ...
3. return output\_shapes

* input\_shapes 和 output\_shapes 是元组列表，用来表示对应的输入/输出的shape。

note：不提供 infer\_shape方法是为了防止对这个op的相关的shape优化 。例如， your\_op(inputs, ...).shape 会需要执行该op从而得到shape。

note：没有grad被定义，也就是不能对包含该op的路径中的函数进行微分。

note：它将python函数转换成可调用的对象，该对象的输入是已经声明过的theano变量。

#### 7.1 as\_op 例子

1. [复制代码](javascript:void(0);)
2. import theano
3. import numpy
4. from theano.compile.ops import as\_op
6. def infer\_shape\_numpy\_dot(node, input\_shapes):
7. ashp, bshp = input\_shapes
8. return [ashp[:-1] + bshp[-1:]]
10. @as\_op(itypes=[theano.tensor.fmatrix, theano.tensor.fmatrix],
11. otypes=[theano.tensor.fmatrix], infer\_shape=infer\_shape\_numpy\_dot)
12. def numpy\_dot(a, b):

return numpy.dot(a, b)

[复制代码](javascript:void(0);)

如下进行测试：

1. x = theano.tensor.fmatrix()
2. y = theano.tensor.fmatrix()
3. f = function([x, y], numpy\_dot(x, y))
4. inp1 = numpy.random.rand(5, 4)
5. inp2 = numpy.random.rand(4, 7)
6. out = f(inp1, inp2)

### 7.2 练习

    运行上的numpy\_dot 例子的代码。

    修改并执行： numpy.add 和 numpy.subtract.

    修改并执行： x + y和 x - y.

### 八、测试中的随机数

    测试错误可重复在实际中是非常具有说服力的，为了让测试可以重复，就需要得到相同的随机树。你可以通过对Numpy的随机数生成器传入相同的种子来得到。

    更方便的方法就是使用类 InferShapeTester 和 RopLop\_checker 。如果你实现了自己的 setUp 函数，别忘了调用其父类的setUp函数。

更多信息请查阅： [Using Random Values in Test Cases](http://deeplearning.net/software/theano/extending/unittest.html#random-value-in-tests).

答案（[Solution](http://deeplearning.net/software/theano/_downloads/extending_theano_solution_1.py)）

1. http://images.cnblogs.com/OutliningIndicators/ContractedBlock.gif View Code

### 九、文档

[Documentation Documentation AKA Meta-Documentation](http://deeplearning.net/software/theano/internal/metadocumentation.html#metadocumentation)中更详细的介绍了如何生成文档。

下面这个例子介绍如何将docstring加到一个类中：

1. [复制代码](javascript:void(0);)
2. import theano
4. class DoubleOp(theano.Op):
5. """ Double each element of a tensor.
7. :param x: input tensor.
9. :return: a tensor of the same shape and dtype as the input with all
10. values doubled.
12. :note:
13. this is a test note
15. :seealso:
16. You can use the elemwise op to replace this example.
17. Just execute `x \* 2` with x being a Theano variable.
19. .. versionadded:: 0.6

"""

[复制代码](javascript:void(0);)

下面就是在库文档中的说明形式：

class theano.misc.doubleop.DoubleOp(use\_c\_code='/usr/bin/g++')

Double each element of a tensor.

|  |  |
| --- | --- |
| **Parameters:** | **x** – input tensor. |
| **Returns:** | a tensor of the same shape and dtype as the input with all values doubled. |
| **Note :** | this is a test note |
| **Seealso :** | You can use the elemwise op to replace this example. Just execute x \* 2 with x being a Theano variable. |

|  |
| --- |
|  |

在0.6版本中新加入的特性。

### 十、最后的笔记

    这部分更深的讨论可以查阅高级教程： [Extending Theano](http://deeplearning.net/software/theano/extending/index.html#extending).

[Other ops](http://deeplearning.net/software/theano/extending/other_ops.html#other-ops) 介绍了下面这几个特殊情况：

* [Scalar/Elemwise/Reduction Ops](http://deeplearning.net/software/theano/extending/other_ops.html#scalar-ops)
* [SciPy Ops](http://deeplearning.net/software/theano/extending/other_ops.html#scipy-ops)
* [Sparse Ops](http://deeplearning.net/software/theano/extending/other_ops.html#sparse-ops)
* [Random ops](http://deeplearning.net/software/theano/extending/other_ops.html#random-ops)
* [OpenMP Ops](http://deeplearning.net/software/theano/extending/other_ops.html#openmp-ops)
* [Numba Ops](http://deeplearning.net/software/theano/extending/other_ops.html#numba-ops)

参考资料：

[1]官网：http://deeplearning.net/software/theano/tutorial/extending\_theano.html

## [Theano2.1.21-基础知识之theano中多核的支持](http://www.cnblogs.com/shouhuxianjian/p/4594524.html)

来自：http://deeplearning.net/software/theano/tutorial/multi\_cores.html

Multi cores support in Theano

# 一、BLAS 操作

    BLAS是一个针对于某些数学上操作的接口，其中包括两个向量、一个向量和一个矩阵、两个矩阵（即，介于向量/矩阵和矩阵/矩阵之间的点积操作）。当然有许多这样接口的不同的实现方法，有些还是可以并行的。

    Theano是想尽可能使用这个接口来保证整体的性能。所以如果想要实现theano的并行，这些操作将会在thenao中并行的运行。

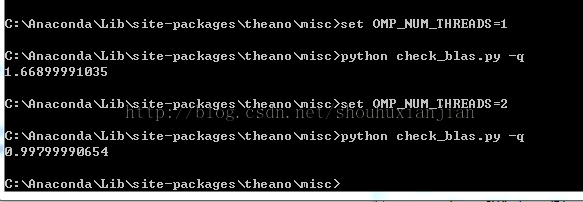
    最频繁的来控制线程的数量的方法就是通过 OMP\_NUM\_THREADS 环境变量来设置。在开启python进程之前，将它设置成你想要使用到的线程的数量。一些BLAS实现也支持其他环境变量。

    为了测试你的BLAS是否支持 OpenMP/Multiple 多核，你可以使用 theano/misc/check\_blas.py 脚本来测试，命令如下：

OMP\_NUM\_THREADS=1 python theano/misc/check\_blas.py -q

OMP\_NUM\_THREADS=2 python theano/misc/check\_blas.py -q

ps：在windows的cmd上运行的时候，需要用到set OMP\_NUM\_THREADS=2。（set会临时的定义环境变量，当关闭一个cmd的时候，就会被清除的。）



个人：待分析

# 二、使用OpenMP来并行的逐元素ops

    因为在每个张量元素上独立的进行逐元素操作可以很容易的通过OpenMP来并行化。

   为了使用OpenMP，你需要设置 openmp [flag](http://deeplearning.net/software/theano/library/config.html#libdoc-config) 为 True.

    你可以使用flag openmp\_elemwise\_minsize 来设置需要并行的最小的张量size。对于较短的张量来说，使用OpenMP反而可能会减慢该操作。默认的值是20w 。

    对于简单的（快速）的操作来说，你可以在非常大的张量得到一个加速；而对于复杂的操作来说，你可以在较小的张量上得到一个不错的加速。

    在theano/misc路径下有一个脚本 elemwise\_openmp\_speedup.py ，你可以用来针对你你的机器来调整 openmp\_elemwise\_minsize 的值。该脚本是对具有size为 openmp\_elemwise\_minsize的向量执行两个逐元素操作（一个快速操作，一个慢速操作） 并分别使用OpenMP和未使用两种情况下执行，从而来显示不同情况下的耗时情况。

    唯一的用来控制线程数量的方法是通过 OMP\_NUM\_THREADS 环境变量。在开启python进程之前将它设置成你想要的线程数量。你可以用下面的命令来测试：

OMP\_NUM\_THREADS=2 python theano/misc/elemwise\_openmp\_speedup.py

#The output

Fast op time without openmp 0.000533s with openmp 0.000474s speedup 1.12

Slow op time without openmp 0.002987s with openmp 0.001553s speedup 1.92

个人：上面的没成功，错误待分析，所以没图。

参考资料：

[1]：官网：http://deeplearning.net/software/theano/tutorial/multi\_cores.html

[**Theano3.1-练习之初步介绍**](http://www.cnblogs.com/shouhuxianjian/p/4564623.html)

        来自 http://deeplearning.net/tutorial/，虽然比较老了，不过觉得想系统的学习theano，所以需要从python--numpy--theano的顺序学习。这里的资料都很老了，毕竟看得出来应该是10年的，现在都15年了，综述还是什么都很不一样了，不过对于学习theano来说，还好没差。如果是想学习dl的原理的，推荐看其他的新综述和文章，这里的博文主要是为了学习theano，也就是主要工程上，不是学术上。

        深度学习是机器学习研究中一个新的领域，是为了朝着机器学习的初衷：人工智能而前进的一步，可以看看这两个笔记： a [brief introduction to Machine Learning for AI](http://www.iro.umontreal.ca/~pift6266/H10/notes/mlintro.html)（[翻译：](http://blog.csdn.net/shouhuxianjian/article/details/46357753)）和an [introduction to Deep Learning algorithms](http://www.iro.umontreal.ca/~pift6266/H10/notes/deepintro.html).（翻译：待加上）

        深度学习是关于从例如图像、声音、文本中学习多层表征和抽象语义。对于更多的dl算法，可以看看下面的资料（虽然觉得老了，都好多年了，不过知识总是不会过时的，只是场景应用不同）：

* 专著或综述性文章 [Learning Deep Architectures for AI](http://www.iro.umontreal.ca/~lisa/publications2/index.php/publications/show/239) (Foundations & Trends in Machine Learning, 2009).
* The ICML 2009 Workshop on Learning Feature Hierarchies [webpage](http://www.cs.toronto.edu/~rsalakhu/deeplearning/index.html) has a [list of references](http://www.cs.toronto.edu/~rsalakhu/deeplearning/references.html).
* The LISA [public wiki](http://www.iro.umontreal.ca/~lisa/twiki/bin/view.cgi/Public/WebHome) has a [reading list](http://www.iro.umontreal.ca/~lisa/twiki/bin/view.cgi/Public/ReadingOnDeepNetworks) and a [bibliography](http://www.iro.umontreal.ca/~lisa/twiki/bin/view.cgi/Public/DeepNetworksBibliography).（读书顺序，推荐，可惜比较旧了）
* Geoff Hinton has [readings](http://www.cs.toronto.edu/~hinton/deeprefs.html) from 2009’s [NIPS tutorial](http://videolectures.net/jul09_hinton_deeplearn/).

        该教程将会介绍一些最重要的dl算法并且会呈现如何使用 [Theano](http://deeplearning.net/software/theano).来运行的。Theano是一个python库，可以用来很容易的写出dl的模型，并且能够在GPU上训练。不过这些算法教程却需要一些先决条件。你需要知道python，很熟悉numpy。而且需要首先读完 [Theano basic tutorial](http://deeplearning.net/software/theano/tutorial) [翻译：](http://blog.csdn.net/shouhuxianjian/article/details/46422627#t2)。一旦你做完了上面这些，就可以读这部分章节了 [*Getting Started*](http://deeplearning.net/tutorial/gettingstarted.html#gettingstarted)–（[翻译：](http://blog.csdn.net/shouhuxianjian/article/details/46375711)）这个章节介绍了一些符号注释，和在这个教程中使用的可下载的数据集 ，这里是通过使用随机梯度下降法（stochastic gradient descent，SGD）来优化的。

下面这几个是纯有监督学习算法，可以作为初始的练习，按照指定顺序做:

1. [*Logistic Regression*](http://deeplearning.net/tutorial/logreg.html#logreg) -[翻译：](http://blog.csdn.net/shouhuxianjian/article/details/46375461)
2. [*Multilayer perceptron*](http://deeplearning.net/tutorial/mlp.html#mlp) -[翻译：](http://blog.csdn.net/shouhuxianjian/article/details/46400947)introduction to layers
3. [*Deep Convolutional Network*](http://deeplearning.net/tutorial/lenet.html#lenet) -[翻译：](http://blog.csdn.net/shouhuxianjian/article/details/46408167)a simplified version of LeNet5

下面这几个是无监督和半监督学习算法，可以按照你想要的顺序做（auto-encoders可以和RBM/DBN独立分开）:

* [*Auto Encoders, Denoising Autoencoders*](http://deeplearning.net/tutorial/dA.html#daa) - description of autoencoders
* [*Stacked Denoising Auto-Encoders*](http://deeplearning.net/tutorial/SdA.html#sda) - easy steps into unsupervised pre-training for deep nets
* [*Restricted Boltzmann Machines*](http://deeplearning.net/tutorial/rbm.html#rbm) - single layer generative RBM model
* [*Deep Belief Networks*](http://deeplearning.net/tutorial/DBN.html#dbn) - unsupervised generative pre-training of stacked RBMs followed by supervised fine-tuning

接下来是需要的mcRBM,这里是一个从能量模型上采样的一个新教程：

* [*HMC Sampling*](http://deeplearning.net/tutorial/hmc.html#hmc) - hybrid (aka Hamiltonian) Monte-Carlo sampling with scan()

接下来的是对比自动编码器的教程，现在已经提供代码了：

* [Contractive auto-encoders](https://github.com/lisa-lab/DeepLearningTutorials/blob/master/code/cA.py) code - There is some basic doc in the code.

接下来的是基于单词嵌入和上下文窗口的递归神经网络（Recurrent neural networks，RNN)：

* [*Semantic Parsing of Speech using Recurrent Net*](http://deeplearning.net/tutorial/rnnslu.html#rnnslu)

情绪分析的LSTM 网络:

* [*LSTM network*](http://deeplearning.net/tutorial/lstm.html#lstm)

基于能量的递归神经网络 (RNN-RBM):

* [*Modeling and generating sequences of polyphonic music*](http://deeplearning.net/tutorial/rnnrbm.html#rnnrbm)

这个博文可以当成一个学习Theano的总入口，接下来会不断的更新其中的原理和代码解析，尽可能的不太监，而且我是先一直更新python的，然后更新了一篇numpy的，然后在来theano的部分的，尽量做到从零开始。

[**Theano3.2-练习之数据集及目标函数介绍**](http://www.cnblogs.com/shouhuxianjian/p/4564627.html)

来自http://deeplearning.net/tutorial/gettingstarted.html#gettingstarted

**一、下载**

   在后续的每个学习算法上，都需要下载对应的文档，如果想要一次全部下好，那么可以复制git上面的这个教程的资料：

git clone git://github.com/lisa-lab/DeepLearningTutorials.git

**二、数据集**

  MNIST 数据集([mnist.pkl.gz](http://deeplearning.net/data/mnist/mnist.pkl.gz))（现在这个数据集除了教学，好像已经没什么人关注了）

    这个[MNIST](http://yann.lecun.com/exdb/mnist) 数据集包含的是手写数字图像，其中有6w张训练样本和1w张测试样本，不过在几乎许多论文和本教程中，都是将这6w张训练样本划分成5w张训练样本和1w张验证集样本，所有的图片都已经中心化而且是固定的大小28×28，其中是灰度图，白色为255，黑色为0.为了方面本教程，是需要在python下使用的，可以下载 [here](http://deeplearning.net/data/mnist/mnist.pkl.gz).也就是已经划分好了三个list：训练集、验证集和测试集。每一个list都是由图像和相应的标签组成的。其中图像是numpy的784（28\*28）的一维数组（就是把2维的图像拉成一条向量），标签则是一个0-9之间的数字。下面的代码演示了如何使用这个数据集：

1. **import** cPickle, gzip, numpy
3. # Load the dataset
4. f = gzip.open('mnist.pkl.gz', 'rb')
5. train\_set, valid\_set, test\_set = cPickle.load(f)
6. f.close()

        当使用这个数据集的时候，通常是将它划分成minibatches（ [*Stochastic Gradient Descent*](http://deeplearning.net/tutorial/gettingstarted.html#opt-sgd)）（还有http://blog.csdn.net/shouhuxianjian/article/details/41040245，这是hinton的视频的第6课）。我们建议你可以将这个数据集放入到共享变量（shared variables）中，并通过基于minibatch索引、给定一个固定和已知的batch size来访问它。这样做的好处就是共享变量之后可以用在gpu上，因为当将数据复制到GPU内存上的时候会有较大的开销，如果按照代码的执行（每个minibatch都是独立的）来进行传输数据的时候，如果不是用共享变量的方法，结果反而比只使用CPU的速度还慢。如果使用Theano 共享变量，那么就是让Theano将整个数据在共享变量构造的时候通过一个单一的调用都复制到GPU上。之后，GPU可以通过在这个共享变量上进行切片slice来访问任何minibatch，而不需要从cpu的内存上复制到GPU上，所以避免了很多的数据传输的开销。因为这些数据点和他们的标签通常都是不同的（标签通常是整数，而数据点通常是实数），我们建议使用不同的变量来表示数据和标签。同样我们推荐使用对这三个不同的集合也采用不同的变量来使得代码更具有可读性（会生成6个不同的共享变量）。因为当前的数据是一个变量，而且一个minibatch可以被定为这个变量的一个切片，所以很自然的可以通过指定索引和尺寸来定义一个minibatch。在我们的步骤中，batch size在代码执行过程中 一直是一个常量，所以一个函数实际上需要的只是索引来指定哪个数据点被使用了。下面的代码来表示如何存储数据并且如何访问一个minibatch：

1. **def** shared\_dataset(data\_xy):
2. """ Function that loads the dataset into shared variables
4. The reason we store our dataset in shared variables is to allow
5. Theano to copy it into the GPU memory (when code is run on GPU).
6. Since copying data into the GPU is slow, copying a minibatch everytime
7. is needed (the default behaviour if the data is not in a shared
8. variable) would lead to a large decrease in performance.
9. """
10. data\_x, data\_y = data\_xy
11. shared\_x = theano.shared(numpy.asarray(data\_x, dtype=theano.config.floatX))#因为GPU只接受float类型
12. shared\_y = theano.shared(numpy.asarray(data\_y, dtype=theano.config.floatX))#因为GPU只接受float类型
13. # When storing data on the GPU it has to be stored as floats
14. # therefore we will store the labels as ``floatX`` as well
15. # (``shared\_y`` does exactly that). But during our computations
16. # we need them as ints (we use labels as index, and if they are
17. # floats it doesn't make sense) therefore instead of returning
18. # ``shared\_y`` we will have to cast it to int. This little hack
19. # lets us get around this issue
20. **return** shared\_x, T.cast(shared\_y, 'int32') #返回的时候强制标签为int类型
22. test\_set\_x, test\_set\_y = shared\_dataset(test\_set)
23. valid\_set\_x, valid\_set\_y = shared\_dataset(valid\_set)
24. train\_set\_x, train\_set\_y = shared\_dataset(train\_set)
26. batch\_size = 500    # size of the minibatch
28. # accessing the third minibatch of the training set 访问训练集的第三个minibatch
30. data  = train\_set\_x[2 \* 500: 3 \* 500]
31. label = train\_set\_y[2 \* 500: 3 \* 500]

        在GPU上存储的数据只能是floats类型的，（存储在GPu上时，右边的dtype被赋值为theano.config.floatX）.为了绕过这个标签上的问题，通过将其存储为float，然后返回的时候强制为int类型。

note：如果你想要在GPU上运行代码，而你使用的数据集太大而无法放入GPU内存中，这种情况下，你可能会将数据存储到一个共享变量中。然而你可以存储一个足够小的数据块（几个minibatches）放到一个共享变量中，然后使用这个来进行训练，当这个数据块训练完成之后，更新存储的数据块换下一部分。这个方法是为了最小化CPU和GPU之间数据传输的次数的折衷方法。

**三、符号**

**数据集符号**

    我们将数据集表示成http://img.blog.csdn.net/20150606094721846?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvc2hvdWh1eGlhbmppYW4=/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/Center，当需要区别对待的时候，将训练集，验证集和测试集表示成：: \mathcal{D}_{train}, \mathcal{D}_{valid} 和\mathcal{D}_{test}。验证集是用来执行模型选取和超参数选择的，而且测试集是用来验证最后的 泛化误差和以无偏的方式来对比不同的算法。该教程基本上处理的是分类问题，这里每个数据集http://img.blog.csdn.net/20150606094721846?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvc2hvdWh1eGlhbmppYW4=/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/Center是有关 (x^{(i)},y^{(i)})对的索引集合，使用上标来区分不同的训练集样本：x^{(i)} \in\mathcal{R}^D 是第 i 个维度为http://img.blog.csdn.net/20150606094721846?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvc2hvdWh1eGlhbmppYW4=/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/Center的训练样本。相似的，y^{(i)} \in \{0, ..., L\}是第 i 个指派给输入x^{(i)}的标签。这样就可以简单的扩展这些例子，使得y^{(i)}能够有其他类型（例如，高斯回归，或者能够预测多个符号的多项式组（也就是多分类））。

**数学约定**

* W: 大写符号用来表示一个矩阵，除非有其他的特指
* W_{ij}: 矩阵W的第i 行第j 列的元素
* W_{i \cdot}, W_i: 向量, 表示矩阵W第i 行
* W_{\cdot j}: 向量, 表示矩阵W第j 列
* b: 小写符号用来表示一个向量，除非有其他的特指
* b_i: 向量b的第i 个元素

**符号和函数的列表**

* D: 输入的维度数.
* D_h^{(i)}: 第i 层隐藏单元的个数.
* f_{\theta}(x), f(x): 和模型 P(Y|x,\theta)相关的分类函数, 定义成 {\rm argmax}_k P(Y=k|x,\theta). 注意到这里通常将下标 \theta丢弃掉.
* L:标签的个数.
* \mathcal{L}(\theta, \cal{D}): 由参数\theta定义模型 的\cal{D}的log似然 .
* \ell(\theta, \cal{D}) 在数据集\cal{D}上由参数\theta决定的预测函数 f 的期望损失.
* NLL: 负似然log（negative log-likelihood，NLL）
* \theta: 对于一个给定模型的所有参数集合

**python的命名空间**本教程的代码通常使用下面的命名空间：

1. **import** theano
2. **import** theano.tensor as T
3. **import** numpy

**四、在DL上的有监督优化的入门**

    深度学习最让人兴奋的主要是利用深网络的无监督学习。不过有监督学习同样也扮演着很重要的角色。无监督预训练的使用通常是基于当使用有监督微调之后获得效果的基础上来进行评估的。本部分回顾下作为分类模型的有监督的基础部分，然后介绍下作为在许多dl教程中模型上使用的微调的minibatch随机梯度下降算法。详细的可以看看[introductory course notes on gradient-based learning](http://www.iro.umontreal.ca/~pift6266/H10/notes/gradient.html) ，来了解更多关于使用梯度来优化训练标准的基本概念。

**学习一个分类器**

**zero-one 损失**

    本教程中呈现的模型大多是用来做分类的。训练一个分类器的目标是为了最小化在不可见样本上的误差（0-1损失）的数量。如果 f: R^D \rightarrow\{0,...,L\}是预测函数，那么损失函数可以写成：

\ell_{0,1} = \sum_{i=0}^{|\mathcal{D}|} I_{f(x^{(i)}) \neq y^{(i)}}

这里是\mathcal{D} 表示的是训练集合（在训练的时候）或者 \mathcal{D} \cap \mathcal{D}_{train} = \emptyset（为了避免验证集或者测试误差的有偏估计）。I 是指示函数，可以被定义成：

I_x = \left\{\begin{array}{ccc}      1&\mbox{ if $x$ is True} \\      0&\mbox{ otherwise}\end{array}\right.

在这个教程中，f 被定义成：

f(x) = {\rm argmax}_k P(Y=k | x, \theta)

在python中，使用Theano的话，可以写成如下形式：

1. # zero\_one\_loss is a Theano variable representing a symbolic
2. # expression of the zero one loss ; to get the actual value this
3. # symbolic expression has to be compiled into a Theano function (see
4. # the Theano tutorial for more details)
5. zero\_one\_loss = T.sum(T.neq(T.argmax(p\_y\_given\_x), y))

**负log似然损失**

    因为0-1损失不能微分，对于大型模型（成千上百万的参数）的优化来说，代价是非常高昂的（计算量）。所以我们在训练集给定所有的标签的基础上让我们的分类器的log似然最大化：

\mathcal{L}(\theta, \mathcal{D}) =    \sum_{i=0}^{|\mathcal{D}|} \log P(Y=y^{(i)} | x^{(i)}, \theta)

    正确类别的似然和正确预测的数量是不相同的，不过从一个随机初始化的分类器的观点上看，它们相当接近。不过提醒下，0-1损失和似然是不同的目标；你需要看见它们是在验证集上是正相关的，不过有时候却是负相关的。因为我们通常说要最小化一个损失函数，所以学习其实就是为了最小化这个负log似然函数，定义为：

NLL(\theta, \mathcal{D}) = - \sum_{i=0}^{|\mathcal{D}|} \log P(Y=y^{(i)} | x^{(i)}, \theta)

我们分类器的NLL是可微分的，所以可以用来代替0-1损失，而且我们在基于训练数据上使用这个函数的梯度作为 一个分类器的dl 有监督学习信号（其实就是用梯度来训练分类器的意思，我直译的而已）。这可以通过使用下面的代码来计算得到：

1. # NLL is a symbolic variable ; to get the actual value of NLL, this symbolic
2. # expression has to be compiled into a Theano function (see the Theano
3. # tutorial for more details)
4. NLL = -T.sum(T.log(p\_y\_given\_x)[T.arange(y.shape[0]), y])
5. # note on syntax: T.arange(y.shape[0]) is a vector of integers [0,1,2,...,len(y)].
6. # Indexing a matrix M by the two vectors [0,1,...,K], [a,b,...,k] returns the
7. # elements M[0,a], M[1,b], ..., M[K,k] as a vector.  Here, we use this
8. # syntax to retrieve the log-probability of the correct labels, y.

**随机梯度下降**

    普通的梯度下降是什么？它是一个简单的算法，在这个算法中首先有由一些参数定义的损失函数表示的错误表面，然后在这个表面上重复的使用很小的步长进行下降的算法。针对于普通的梯度下降法的目的来说，训练数据是需要放入到这个损失函数中的。然后这个算法的伪代码可以写成如下形式：

1. # GRADIENT DESCENT
3. **while** True:
4. loss = f(params)
5. d\_loss\_wrt\_params = ... # compute gradient
6. params -= learning\_rate \* d\_loss\_wrt\_params
7. **if** <stopping condition **is** met>:
8. **return** params

    随机梯度下降法是和用普通梯度下降法一样的原则来work的，不过可以通过每次一点样本来计算梯度从而更快速的进行处理，所以不需要一次放入整个训练样本了。对应的伪代码如下：

1. # STOCHASTIC GRADIENT DESCENT
2. **for** (x\_i,y\_i) **in** training\_set:
3. # imagine an infinite generator
4. # that may repeat examples (if there is only a finite training set)
5. loss = f(params, x\_i, y\_i)
6. d\_loss\_wrt\_params = ... # compute gradient
7. params -= learning\_rate \* d\_loss\_wrt\_params
8. **if** <stopping condition **is** met>:
9. **return** params

     在dl 上我们推荐使用在随机梯度上的进一步变体，叫做“minibatches”。minibatch sgd的工作规则是和sgd一样的，只是我们在每次的梯度估计上使用不止一个训练样本来训练。这个技术可以梯度估计中间的方差，而且通常在现代计算机中可以更好地利用层级存储的组织方式：

1. **for** (x\_batch,y\_batch) **in** train\_batches:
2. # imagine an infinite generator
3. # that may repeat examples
4. loss = f(params, x\_batch, y\_batch)
5. d\_loss\_wrt\_params = ... # compute gradient using theano
6. params -= learning\_rate \* d\_loss\_wrt\_params
7. **if** <stopping condition **is** met>:
8. **return** params

    这是在minibatch size  B的选择上的权衡考虑。方差的减小和SIMD指令的使用在当B从1增加到2的时候通常是很有帮助的，不过这个很小的提升却会很快的回归虚无。使用更大的B，时间会消耗在减少梯度估计器的方差减少上，本来这些时间是应该更好的用在额外的梯度步长上的。一个最优的B是基于模型、数据集、和硬件考虑的，同时可以在任何地方从1上升到甚至好几百。在这个教程中，我们将它设置成20，不过这个选择通常是任意的。  
note：如果你训练的时候使用的是固定数量的epochs，那么这个minibatch size就变得很重要了，因为它控制着你的参数的更新次数。使用batch size 为1 的10次epochs来训练相同的模型得到的结果完全不同于训练batch size 为20的而且也是10个epochs的结果。记得，在不同的batch sizes之间转换的时候，记得按照使用过的这个batch size 来调整所有的其他参数。

    上面所有的演示该算法的伪代码块，在theano中执行同样的算法的代码如下：

1. # Minibatch Stochastic Gradient Descent
3. # assume loss is a symbolic description of the loss function given
4. # the symbolic variables params (shared variable), x\_batch, y\_batch;
6. # compute gradient of loss with respect to params
7. d\_loss\_wrt\_params = T.grad(loss, params)
9. # compile the MSGD step into a theano function
10. updates = [(params, params - learning\_rate \* d\_loss\_wrt\_params)]
11. MSGD = theano.function([x\_batch,y\_batch], loss, updates=updates)
13. **for** (x\_batch, y\_batch) **in** train\_batches:
14. # here x\_batch and y\_batch are elements of train\_batches and
15. # therefore numpy arrays; function MSGD also updates the params
16. **print**('Current loss is ', MSGD(x\_batch, y\_batch))
17. **if** stopping\_condition\_is\_met:
18. **return** params

**正则化**

    除了优化，在机器学习中还有更重要的部分。当我们从数据中训练我们的模型的时候，我们是将它准备用在新样本上的，而不是那些我们已经见过的样本。上面的MSGD的训练循环如果没有考虑到这一点，也许就会过拟合训练样本。一个对应过你的方法就是正则化。这里有好几种正则化的方法，这里会介绍L1/L2正则化和早期停止。

**L1/L2 正则化**    L2和L2正则化涉及到在损失函数上增加额外的项，用来惩罚某一个参数组合。形式上，如果我们的损失函数是：

NLL(\theta, \mathcal{D}) = - \sum_{i=0}^{|\mathcal{D}|} \log P(Y=y^{(i)} | x^{(i)}, \theta)

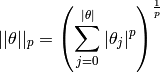
那么正则化损失就该是：

E(\theta, \mathcal{D}) =  NLL(\theta, \mathcal{D}) + \lambda R(\theta)\\

或者，在我们的情况中：

E(\theta, \mathcal{D}) =  NLL(\theta, \mathcal{D}) + \lambda||\theta||_p^p

这里



这是 \theta的L_p 范数。\lambda是一个超参数，用来控制正则化参数的相关重要性。通常 p的值是1和2，所以命名为L1/L2，如果p=2，那么这个正则化叫做“权重衰减”。原则上说，对损失函数增加一个正则化项将会使得在NN中网络更加的平滑（通过惩罚值较大的参数，这些值较大的参数会降低网络模型的非线性程度，所以需要惩罚）。更直观的说，这两项（NLL和R(\theta)）对应于很好的对数据进行建模（NLL）和有着“简单”或“平滑”的解决方法R(\theta)。因此，最小化这两个项的和，从理论上来说，就是为了在拟合训练数据和解决方法的“泛化”之间找到正确的平衡点。为了遵循Occam的razor原则，这个最小化应该让我们找到最简单的解决方法（通过我们简单的标准来测量的）来拟合训练数据。注意到这样一个事实，一个所谓的“简单”的解决方法不是意味着能够很好的泛化。经验上来说，通常是在NN的背景下这样的正则化的添加有助于泛化，特别是对于小的数据集来说。下面的代码块用来表示当包含由\lambda_1来权重化的L1正则化项和由\lambda_2来权重化的L2正则化项的时候如何在python中计算损失的：

1. # symbolic Theano variable that represents the L1 regularization term
2. L1  = T.sum(abs(param))
4. # symbolic Theano variable that represents the squared L2 term
5. L2\_sqr = T.sum(param \*\* 2)
7. # the loss
8. loss = NLL + lambda\_1 \* L1 + lambda\_2 \* L2

**早期停止**

    用早期停止来解决过拟合是通过在验证集合上监测模型的执行结果来完成的。验证集就是我们在梯度下降的时候未使用的样本集，不过这同样也不是测试集的一部分。验证集样本是被认为作为未来测试集样本的代表的。我们可以在训练的时候使用时因为它们不是测试集的一部分。如果模型的效果在验证集上已经停止了提升，或者甚至在后面的优化上还有下降，那么这里需要做的就是停止使用更多的优化。选择什么时候停止是一个主观判断而且是存在启发式的，不过这些教程将会在基于会具有几何增长的patience数量上使用一些策略：

1. # early-stopping parameters
2. patience = 5000  # look as this many examples regardless
3. patience\_increase = 2     # wait this much longer when a new best is
4. # found
5. improvement\_threshold = 0.995  # a relative improvement of this much is
6. # considered significant
7. validation\_frequency = min(n\_train\_batches, patience/2)
8. # go through this many
9. # minibatches before checking the network
10. # on the validation set; in this case we
11. # check every epoch
13. best\_params = None
14. best\_validation\_loss = numpy.inf
15. test\_score = 0.
16. start\_time = time.clock()
18. done\_looping = False
19. epoch = 0
20. **while** (epoch < n\_epochs) **and** (**not** done\_looping):
21. # Report "1" for first epoch, "n\_epochs" for last epoch
22. epoch = epoch + 1
23. **for** minibatch\_index **in** xrange(n\_train\_batches):
25. d\_loss\_wrt\_params = ... # compute gradient
26. params -= learning\_rate \* d\_loss\_wrt\_params # gradient descent
28. # iteration number. We want it to start at 0.
29. iter = (epoch - 1) \* n\_train\_batches + minibatch\_index
30. # note that if we do `iter % validation\_frequency` it will be
31. # true for iter = 0 which we do not want. We want it true for
32. # iter = validation\_frequency - 1.
33. **if** (iter + 1) % validation\_frequency == 0:
35. this\_validation\_loss = ... # compute zero-one loss on validation set
37. **if** this\_validation\_loss < best\_validation\_loss:
39. # improve patience if loss improvement is good enough
40. **if** this\_validation\_loss < best\_validation\_loss \* improvement\_threshold:
42. patience = max(patience, iter \* patience\_increase)
43. best\_params = copy.deepcopy(params)
44. best\_validation\_loss = this\_validation\_loss
46. **if** patience <= iter:
47. done\_looping = True
48. **break**
50. # POSTCONDITION:
51. # best\_params refers to the best out-of-sample parameters observed during the optimization

    如果我们在跑完patience之前跑完了所有的训练数据，那么我们只需要回到训练数据的开始部分，然后再来一次。

note：validation\_frequency应该总是要小于patience的。在跑完patience之前代码需要检查至少两次。这是因为我们使用的公式validation\_frequency = min( value,patience/2)。

note：当决定什么时候需要增大patience的时候，算法可以通过使用统计测试的方法来明显的提升，而不是简单的使用对比。

**测试**

    在现有的循环之后，best\_params变量表示在验证集上best-performing的模型。如果我们给另一个模型类别重复这个过程，或者甚至使用另一个随机初始化，我们应该也要对数据使用相同的train/valid/test划分，然后得到其他best-performing模型。如果我们不得不需要选择最好的模型类别或者最好的初始化，我们需要对每个模型进行对比best\_validation\_loss。当我们选择我们认为的最好的模型（基于验证集）的时候，我们会将这个模型用在测试集上，并报告结果。

**回顾**

    这是为了优化部分准备的。早期停止的技术需要我们将样本集合划分成三个不同的集合（训练集、验证集、测试集）。训练集用来作为目标函数的可微分的近似函数的minibatch sgd上。当我们执行梯度下降的时候，我们定期的使用验证集来观察我们在真正的目标函数上模型的结果（或者至少从经验上分析）。当我们在验证集上看到一个好的模型的时候，我们需要保存下来，当我们发现从看到一个好模型已经过去了很久，那么我们就放弃我们的研究，回头去找到那些最好的参数，然后在测试集上进行评估。

**五、theano/python的提示**

**装载和保存模型**

    当你做实验的时候，会花费好几个小时（或者几天）来做梯度下降然后找到最好的参数。一旦你找到了它们，你将会需要保存这些权重。随着研究的开展，你也许同样会想要保存你当前最好的结果。

**从共享变量中pickle这个numpy ndarrays**

    最好的保存/存档你的模型的参数的方法是使用pickle或者深度复制ndarray对象。例如，如果你的参数都放在共享变量w，v，u中，那么你可以像下面的命令来保存：

1. >>> **import** cPickle
2. >>> save\_file = open('path', 'wb')  # this will overwrite current contents
3. >>> cPickle.dump(w.get\_value(borrow=True), save\_file, -1)  # the -1 is for HIGHEST\_PROTOCOL
4. >>> cPickle.dump(v.get\_value(borrow=True), save\_file, -1)  # .. and it triggers much more efficient
5. >>> cPickle.dump(u.get\_value(borrow=True), save\_file, -1)  # .. storage than numpy's default
6. >>> save\_file.close()

然后，你可以像这样装载你的数据：

1. >>> save\_file = open('path')
2. >>> w.set\_value(cPickle.load(save\_file), borrow=True)
3. >>> v.set\_value(cPickle.load(save\_file), borrow=True)
4. >>> u.set\_value(cPickle.load(save\_file), borrow=True)

这些技术是有一点过于详尽了，不过试过都是正确的。你可以在matplotlib中毫无问题的装载你的数据然后对它进行加工。

**不要为需要长期存储的目的而pickle你的训练或测试函数**

    theano函数是兼容python的深度复制和pickle机制的，不过你没必要一定pickle一个theano函数。如果你更新你的theano文件夹或者说其内部有一些改变，那么你也许没法unpickle你的模型。theano仍然是一个动态的开发项目，内部的APIs可能会改变。所以从安全角度上来说，不要为了长期的存储而pickle你的整个训练或者测试函数（也就是如果save几天或者几个礼拜估计还ok，就是怕几个月之后更新了theano，之前的保存的就没法读取了）。pickle机智是为了短期存储而准备的，例如一个临时文件，或者在一个分布式工作中从一个机器上复制到另一个机器上。

了解更多可以看看 [serialization in Theano](http://deeplearning.net/software/theano/tutorial/loading_and_saving.html), 或者 Python的 [pickling](http://docs.python.org/library/pickle.html).

**显示中间的结果**

    可视化对于理解你的模型或者训练算法在干什么是很有帮助的工具。你可能需要视图键入matplotlib画图命令，或者PIL 图像呈现命令到你的模型训练脚本中。然而，之后你可能想要从这些预呈现的图形中显示一些你感兴趣的或者想要看看从图形中得到的是否清晰，那么你需要保存原始的模型。

**如果你有足够的磁盘空间，你的训练脚本应该保存中间模型并且一个可视化脚本应该用来处理这些保存的模型。**

你已经有了一个模型保存函数了吗？可以再次使用它来保存这些中间模型

你可能会想要了解的库：python图像库 ([PIL](http://www.pythonware.com/products/pil)), [matplotlib](http://matplotlib.sourceforge.net/).

**参考资料：**

[1] 官网：http://deeplearning.net/tutorial/gettingstarted.html#gettingstarted

[2] theano学习指南1：http://www.cnblogs.com/xueliangliu/archive/2013/04/03/2997437.html

[**Theano3.3-练习之逻辑回归**](http://www.cnblogs.com/shouhuxianjian/p/4564626.html)

是官网上theano的逻辑回归的练习（http://deeplearning.net/tutorial/logreg.html#logreg）的讲解。

Classifying MNIST digits using Logistic Regression

note：这部分假设你已经熟悉了这几个theano概念：: [shared variables](http://deeplearning.net/software/theano/tutorial/examples.html#using-shared-variables) , [basic arithmetic ops](http://deeplearning.net/software/theano/tutorial/adding.html#adding-two-scalars) , [T.grad](http://deeplearning.net/software/theano/tutorial/examples.html#computing-gradients) , [floatX](http://deeplearning.net/software/theano/library/config.html#config.floatX).。如果你想要在GPU上运行这个代码，同样可以读读[GPU](http://deeplearning.net/software/theano/tutorial/using_gpu.html).

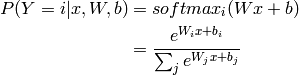
note：这部分的代码下载地址 [here](http://deeplearning.net/tutorial/code/logistic_sgd.py).

    在这部分中，我们会介绍如何使用theano来执行最基础的分类器：逻辑回归。我们先快速的简单介绍下这个模型，这算是一个复习或者说是为了加深理解，并且用来说明如何将数学表达式映射到theano图（graphs）上。在最底层的机器学习传统基础上，本教程来解决MNIST数字分类的最令人激动人心的问题。

**一、模型**

    逻辑回归模型是个概率、线性分类器。它是通过权重矩阵W 和一个偏置向量b来参数化的。通过将输入向量映射到一个超平面集合上来达到分类的目的，每一个超平面都对应着不同的类别。从输入到一个超平面之间的距离反映出该输入是对应的类别成员的概率的大小。

    数学上来说，一个输入向量x 是一个类别 i 的成员的概率是随机变量 Y 的一个值，可以写成如下形式：



模型的预测结果y_{pred} 表示的类别是它的概率是最大的时候，也就是：

y_{pred} = {\rm argmax}_i P(Y=i|x,W,b)

在theano中的代码如下：

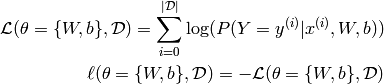
1. # initialize with 0 the weights W as a matrix of shape (n\_in, n\_out)将权重矩阵W的值初始化为0，floatX类别
2. self.W = theano.shared(
3. value=numpy.zeros(
4. (n\_in, n\_out),
5. dtype=theano.config.floatX
6. ),
7. name='W',
8. borrow=True
9. )
10. # initialize the baises b as a vector of n\_out 0s#初始化偏置向量为0，floatX类型
11. self.b = theano.shared(
12. value=numpy.zeros(
13. (n\_out,),
14. dtype=theano.config.floatX
15. ),
16. name='b',
17. borrow=True
18. )
20. # symbolic expression for computing the matrix of class-membership
21. # probabilities
22. # Where:
23. # W is a matrix where column-k represent the separation hyper plain for
24. # class-k
25. # x is a matrix where row-j  represents input training sample-j
26. # b is a vector where element-k represent the free parameter of hyper
27. # plain-k
28. self.p\_y\_given\_x = T.nnet.softmax(T.dot(input, self.W) + self.b)#直接调用softmax计算输入、权重、偏置的结果
30. # symbolic description of how to compute prediction as class whose
31. # probability is maximal
32. self.y\_pred = T.argmax(self.p\_y\_given\_x, axis=1)#读取得到结果的最大值，即softmax得到的最大概率

    因为模型的参数在训练中必须保持一个稳定的状态，我们需要为 W,b分配共享变量。这样不但是将他们声明为符号式的theano变量，而且同样对它们进行了初始化。点和softmax操作随后用来计算向量 P(Y|x,W,b)。结果p\_y\_given\_x是一个向量类型的符号变量。为了得到实际模型的预测结果，可以使用T.argmax操作，这个用来返回在p\_y\_given\_x中的最大值（即最大概率的那个类）。现在，该模型还没有任何用处，因为它的参数仍然在初始化的状态，下面的部分将会介绍如何来学习优化这些参数

note：theano的操作的完整列表：[list of ops](http://deeplearning.net/software/theano/library/tensor/basic.html#basic-tensor-functionality)

**二、定义一个损失函数**

    学习一个最优化模型的参数，需要涉及到最小化一个损失函数。在多类逻辑回归中，通常是使用负log似然作为损失函数。也就是等于在基于由 \theta参数化模型后，最大化数据集\cal{D}的似然函数。让我们首先定义下似然函数\cal{L} 和损失函数\ell：



    然而整本书都在说最小化的主题，到目前为止梯度下降算是最简单的方法用来最小化任意非线性函数了。该教程将会使用minibatch sgd的方法。更详细的见[*Stochastic Gradient Descent*](http://deeplearning.net/tutorial/gettingstarted.html#opt-sgd) 。

下面的theano代码是在给定minibatch的基础上定义的损失（符号）函数：

1. # y.shape[0] is (symbolically) the number of rows in y, i.e.,
2. # number of examples (call it n) in the minibatch
3. # T.arange(y.shape[0]) is a symbolic vector which will contain
4. # [0,1,2,... n-1] T.log(self.p\_y\_given\_x) is a matrix of
5. # Log-Probabilities (call it LP) with one row per example and
6. # one column per class LP[T.arange(y.shape[0]),y] is a vector
7. # v containing [LP[0,y[0]], LP[1,y[1]], LP[2,y[2]], ...,
8. # LP[n-1,y[n-1]]] and T.mean(LP[T.arange(y.shape[0]),y]) is
9. # the mean (across minibatch examples) of the elements in v,
10. # i.e., the mean log-likelihood across the minibatch.
11. **return** -T.mean(T.log(self.p\_y\_given\_x)[T.arange(y.shape[0]), y])

note：即使损失函数在形式上是定义成基于数据集的独立误差项的和。不过在实际操作中，在代码中是通过使用均值（T.mean）来实现的。这使得我们对于学习率的选择能更少的依赖于minibatch size。

**三、创建一个逻辑回归类**

    我们现在有所有我们需要定义一个LogisticRegression类的工具。可以用一个类来封装逻辑回归的基本操作。这里的代码很类似于到目前为止看到的代码，所以可以自圆其说：

1. **class** LogisticRegression(object):
2. """Multi-class Logistic Regression Class
4. The logistic regression is fully described by a weight matrix :math:`W`
5. and bias vector :math:`b`. Classification is done by projecting data
6. points onto a set of hyperplanes, the distance to which is used to
7. determine a class membership probability.
8. """
10. **def** \_\_init\_\_(self, input, n\_in, n\_out):
11. """ Initialize the parameters of the logistic regression
13. :type input: theano.tensor.TensorType
14. :param input: symbolic variable that describes the input of the
15. architecture (one minibatch)
17. :type n\_in: int
18. :param n\_in: number of input units, the dimension of the space in
19. which the datapoints lie
21. :type n\_out: int
22. :param n\_out: number of output units, the dimension of the space in
23. which the labels lie
25. """
26. # start-snippet-1
27. # initialize with 0 the weights W as a matrix of shape (n\_in, n\_out)
28. self.W = theano.shared(
29. value=numpy.zeros(
30. (n\_in, n\_out),
31. dtype=theano.config.floatX
32. ),
33. name='W',
34. borrow=True
35. )
36. # initialize the baises b as a vector of n\_out 0s
37. self.b = theano.shared(
38. value=numpy.zeros(
39. (n\_out,),
40. dtype=theano.config.floatX
41. ),
42. name='b',
43. borrow=True
44. )
46. # symbolic expression for computing the matrix of class-membership
47. # probabilities
48. # Where:
49. # W is a matrix where column-k represent the separation hyper plain for
50. # class-k
51. # x is a matrix where row-j  represents input training sample-j
52. # b is a vector where element-k represent the free parameter of hyper
53. # plain-k
54. self.p\_y\_given\_x = T.nnet.softmax(T.dot(input, self.W) + self.b)
56. # symbolic description of how to compute prediction as class whose
57. # probability is maximal
58. self.y\_pred = T.argmax(self.p\_y\_given\_x, axis=1)
59. # end-snippet-1
61. # parameters of the model
62. self.params = [self.W, self.b]
64. **def** negative\_log\_likelihood(self, y):
65. """Return the mean of the negative log-likelihood of the prediction
66. of this model under a given target distribution.
68. .. math::
70. \frac{1}{|\mathcal{D}|} \mathcal{L} (\theta=\{W,b\}, \mathcal{D}) =
71. \frac{1}{|\mathcal{D}|} \sum\_{i=0}^{|\mathcal{D}|}
72. \log(P(Y=y^{(i)}|x^{(i)}, W,b)) \\
73. \ell (\theta=\{W,b\}, \mathcal{D})
75. :type y: theano.tensor.TensorType
76. :param y: corresponds to a vector that gives for each example the
77. correct label
79. Note: we use the mean instead of the sum so that
80. the learning rate is less dependent on the batch size
81. """
82. # start-snippet-2
83. # y.shape[0] is (symbolically) the number of rows in y, i.e.,
84. # number of examples (call it n) in the minibatch
85. # T.arange(y.shape[0]) is a symbolic vector which will contain
86. # [0,1,2,... n-1] T.log(self.p\_y\_given\_x) is a matrix of
87. # Log-Probabilities (call it LP) with one row per example and
88. # one column per class LP[T.arange(y.shape[0]),y] is a vector
89. # v containing [LP[0,y[0]], LP[1,y[1]], LP[2,y[2]], ...,
90. # LP[n-1,y[n-1]]] and T.mean(LP[T.arange(y.shape[0]),y]) is
91. # the mean (across minibatch examples) of the elements in v,
92. # i.e., the mean log-likelihood across the minibatch.
93. **return** -T.mean(T.log(self.p\_y\_given\_x)[T.arange(y.shape[0]), y])
94. # end-snippet-2
96. **def** errors(self, y):
97. """Return a float representing the number of errors in the minibatch
98. over the total number of examples of the minibatch ; zero one
99. loss over the size of the minibatch
101. :type y: theano.tensor.TensorType
102. :param y: corresponds to a vector that gives for each example the
103. correct label
104. """
106. # check if y has same dimension of y\_pred
107. **if** y.ndim != self.y\_pred.ndim:
108. **raise** TypeError(
109. 'y should have the same shape as self.y\_pred',
110. ('y', y.type, 'y\_pred', self.y\_pred.type)
111. )
112. # check if y is of the correct datatype
113. **if** y.dtype.startswith('int'):
114. # the T.neq operator returns a vector of 0s and 1s, where 1
115. # represents a mistake in prediction
116. **return** T.mean(T.neq(self.y\_pred, y))
117. **else**:
118. **raise** NotImplementedError()

可以通过下面这样实例化这个类：

1. # generate symbolic variables for input (x and y represent a
2. # minibatch)
3. x = T.matrix('x')  # data, presented as rasterized images
4. y = T.ivector('y')  # labels, presented as 1D vector of [int] labels
6. # construct the logistic regression class
7. # Each MNIST image has size 28\*28
8. classifier = LogisticRegression(input=x, n\_in=28 \* 28, n\_out=10)

     我们先对训练输入分配符号变量x和对应的类别标签分配符号变量y。注意到x 和y 都是在LogisticRegression对象范围的外部定义的。因为该类需要输入来建立它的图（graph），所以需要将它作为参数传递给\_\_init\_\_函数。当你想要将这些类的实例连接起来形成一个深度网络的时候是很有用的。每一层的输出可以传递给上一层的输入（该教程不建立一个多层网络，不过该骂却可以在未来教程中重用）。最后，我们定义一个（符号）cost变量来最小化，使用实例方法classifier.negative\_log\_likehood：

1. # the cost we minimize during training is the negative log likelihood of
2. # the model in symbolic format
3. cost = classifier.negative\_log\_likelihood(y)

注意到因为符号变量classifier是定义在以x项为初始化的基础上的，所以x 是一个对于cost的定义是隐式符号输入。

**四、模型的学习**

    为了在大多数编程语言（c/c++、Matlab、Python）中执行MSGD，需要手动计算关于参数的损失函数梯度的表达式：在这种情况下 \partial{\ell}/\partial{W}, 和 \partial{\ell}/\partial{b}，对于复杂模型来说可能需要有相当的技巧，比如表达式\partial{\ell}/\partial{\theta}可以变得相当复杂，特别是还要考虑数值稳定的问题的时候。不过通过theano，这个工作就变得相当简单了，它会自动进行分化（differentiation），然后使用某个数学变换来提升数值稳定性。为了在theano中得到 \partial{\ell}/\partial{W} 和\partial{\ell}/\partial{b} 的梯度，简单的按照下面的操作：

1. g\_W = T.grad(cost=cost, wrt=classifier.W)
2. g\_b = T.grad(cost=cost, wrt=classifier.b)

    g\_w和g\_b都是符号变量，用来作为计算图中的一部分。函数train\_model是用来执行一步梯度的，可以被定义成如下形式：

1. # specify how to update the parameters of the model as a list of
2. # (variable, update expression) pairs.
3. updates = [(classifier.W, classifier.W - learning\_rate \* g\_W),
4. (classifier.b, classifier.b - learning\_rate \* g\_b)]
6. # compiling a Theano function `train\_model` that returns the cost, but in
7. # the same time updates the parameter of the model based on the rules
8. # defined in `updates`
9. train\_model = theano.function(
10. inputs=[index],
11. outputs=cost,
12. updates=updates,
13. givens={
14. x: train\_set\_x[index \* batch\_size: (index + 1) \* batch\_size],
15. y: train\_set\_y[index \* batch\_size: (index + 1) \* batch\_size]
16. }
17. )

updates是一个对列表。在每个对中，第一个元素是符号变量用来在每一步中进行更新，第二个元素是符号函数用来计算新的值。相似的，givens是一个字典，它的键是符号变量，它的值是在每一步中指定的替换值。函数train\_model可以被如下规则定义：

* 输入是minibatch的索引 index 和 batch size (这不是输入，因为它是固定的) 用来定义 x 和对应的标签 y
* 返回值是由索引index定义的x和y 的cost/loss函数值
* 在每次的函数调用上，首先会通过索引index切片的训练集来替换x 和y .然后，通过这个新的minibatch来估算对应的cost函数值，然后使用updates列表中定义的操作.

每次train\_model(index)被调用的时候，它都会计算一个minibatch的cost然后返回，同时执行一步MSGD操作。整个学习算法会在数据集的所有样本上进行循环，考虑到一次只有一个minibatch中的所有样本，需要重复的调用train\_model函数。

**五、测试该模型**

    正如[*Learning a Classifier*](http://deeplearning.net/tutorial/gettingstarted.html#opt-learn-classifier)中说的，当测试这个模型，感兴趣于其中有多少误分类的样本（不止是在似然函数中）。LogisticRegression类会有一个额外的实例方法，用来建立符号图，从而对每个minibatch中误分类样本进行检索。

代码如下:

1. **def** errors(self, y):
2. """Return a float representing the number of errors in the minibatch
3. over the total number of examples of the minibatch ; zero one
4. loss over the size of the minibatch
6. :type y: theano.tensor.TensorType
7. :param y: corresponds to a vector that gives for each example the
8. correct label
9. """
11. # check if y has same dimension of y\_pred
12. **if** y.ndim != self.y\_pred.ndim:
13. **raise** TypeError(
14. 'y should have the same shape as self.y\_pred',
15. ('y', y.type, 'y\_pred', self.y\_pred.type)
16. )
17. # check if y is of the correct datatype
18. **if** y.dtype.startswith('int'):
19. # the T.neq operator returns a vector of 0s and 1s, where 1
20. # represents a mistake in prediction
21. **return** T.mean(T.neq(self.y\_pred, y))
22. **else**:
23. **raise** NotImplementedError()

    然后我们创建一个函数test\_model和一个函数validate\_model。正如你将会看到的，validate\_model是我们早期停止方法中的关键（ [*Early-Stopping*](http://deeplearning.net/tutorial/gettingstarted.html#opt-early-stopping)）。这些函数会将minibatch index作为输入然后计算由模型误分类的序列号。这两个函数之间唯一的区别在于test\_model是从测试集中提取minibatches，而validate\_model是从验证集中提取的：

1. # compiling a Theano function that computes the mistakes that are made by
2. # the model on a minibatch
3. test\_model = theano.function(
4. inputs=[index],
5. outputs=classifier.errors(y),
6. givens={
7. x: test\_set\_x[index \* batch\_size: (index + 1) \* batch\_size],
8. y: test\_set\_y[index \* batch\_size: (index + 1) \* batch\_size]
9. }
10. )
12. validate\_model = theano.function(
13. inputs=[index],
14. outputs=classifier.errors(y),
15. givens={
16. x: valid\_set\_x[index \* batch\_size: (index + 1) \* batch\_size],
17. y: valid\_set\_y[index \* batch\_size: (index + 1) \* batch\_size]
18. }
19. )

**六、合并上面所有的**

完成的成品代码如下：

1. """
2. This tutorial introduces logistic regression using Theano and stochastic
3. gradient descent.
5. Logistic regression is a probabilistic, linear classifier. It is parametrized
6. by a weight matrix :math:`W` and a bias vector :math:`b`. Classification is
7. done by projecting data points onto a set of hyperplanes, the distance to
8. which is used to determine a class membership probability.
10. Mathematically, this can be written as:
12. .. math::
13. P(Y=i|x, W,b) &= softmax\_i(W x + b) \\
14. &= \frac {e^{W\_i x + b\_i}} {\sum\_j e^{W\_j x + b\_j}}

17. The output of the model or prediction is then done by taking the argmax of
18. the vector whose i'th element is P(Y=i|x).
20. .. math::
22. y\_{pred} = argmax\_i P(Y=i|x,W,b)

25. This tutorial presents a stochastic gradient descent optimization method
26. suitable for large datasets.

29. References:
31. - textbooks: "Pattern Recognition and Machine Learning" -
32. Christopher M. Bishop, section 4.3.2
34. """
35. \_\_docformat\_\_ = 'restructedtext en'
37. **import** cPickle
38. **import** gzip
39. **import** os
40. **import** sys
41. **import** time
43. **import** numpy
45. **import** theano
46. **import** theano.tensor as T

49. **class** LogisticRegression(object):
50. """Multi-class Logistic Regression Class
52. The logistic regression is fully described by a weight matrix :math:`W`
53. and bias vector :math:`b`. Classification is done by projecting data
54. points onto a set of hyperplanes, the distance to which is used to
55. determine a class membership probability.
56. """
58. **def** \_\_init\_\_(self, input, n\_in, n\_out):
59. """ Initialize the parameters of the logistic regression
61. :type input: theano.tensor.TensorType
62. :param input: symbolic variable that describes the input of the
63. architecture (one minibatch)
65. :type n\_in: int
66. :param n\_in: number of input units, the dimension of the space in
67. which the datapoints lie
69. :type n\_out: int
70. :param n\_out: number of output units, the dimension of the space in
71. which the labels lie
73. """
74. # start-snippet-1
75. # initialize with 0 the weights W as a matrix of shape (n\_in, n\_out)
76. self.W = theano.shared(
77. value=numpy.zeros(
78. (n\_in, n\_out),
79. dtype=theano.config.floatX
80. ),
81. name='W',
82. borrow=True
83. )
84. # initialize the baises b as a vector of n\_out 0s
85. self.b = theano.shared(
86. value=numpy.zeros(
87. (n\_out,),
88. dtype=theano.config.floatX
89. ),
90. name='b',
91. borrow=True
92. )
94. # symbolic expression for computing the matrix of class-membership
95. # probabilities
96. # Where:
97. # W is a matrix where column-k represent the separation hyper plain for
98. # class-k
99. # x is a matrix where row-j  represents input training sample-j
100. # b is a vector where element-k represent the free parameter of hyper
101. # plain-k
102. self.p\_y\_given\_x = T.nnet.softmax(T.dot(input, self.W) + self.b)
104. # symbolic description of how to compute prediction as class whose
105. # probability is maximal
106. self.y\_pred = T.argmax(self.p\_y\_given\_x, axis=1)
107. # end-snippet-1
109. # parameters of the model
110. self.params = [self.W, self.b]
112. **def** negative\_log\_likelihood(self, y):
113. """Return the mean of the negative log-likelihood of the prediction
114. of this model under a given target distribution.
116. .. math::
118. \frac{1}{|\mathcal{D}|} \mathcal{L} (\theta=\{W,b\}, \mathcal{D}) =
119. \frac{1}{|\mathcal{D}|} \sum\_{i=0}^{|\mathcal{D}|}
120. \log(P(Y=y^{(i)}|x^{(i)}, W,b)) \\
121. \ell (\theta=\{W,b\}, \mathcal{D})
123. :type y: theano.tensor.TensorType
124. :param y: corresponds to a vector that gives for each example the
125. correct label
127. Note: we use the mean instead of the sum so that
128. the learning rate is less dependent on the batch size
129. """
130. # start-snippet-2
131. # y.shape[0] is (symbolically) the number of rows in y, i.e.,
132. # number of examples (call it n) in the minibatch
133. # T.arange(y.shape[0]) is a symbolic vector which will contain
134. # [0,1,2,... n-1] T.log(self.p\_y\_given\_x) is a matrix of
135. # Log-Probabilities (call it LP) with one row per example and
136. # one column per class LP[T.arange(y.shape[0]),y] is a vector
137. # v containing [LP[0,y[0]], LP[1,y[1]], LP[2,y[2]], ...,
138. # LP[n-1,y[n-1]]] and T.mean(LP[T.arange(y.shape[0]),y]) is
139. # the mean (across minibatch examples) of the elements in v,
140. # i.e., the mean log-likelihood across the minibatch.
141. **return** -T.mean(T.log(self.p\_y\_given\_x)[T.arange(y.shape[0]), y])
142. # end-snippet-2
144. **def** errors(self, y):
145. """Return a float representing the number of errors in the minibatch
146. over the total number of examples of the minibatch ; zero one
147. loss over the size of the minibatch
149. :type y: theano.tensor.TensorType
150. :param y: corresponds to a vector that gives for each example the
151. correct label
152. """
154. # check if y has same dimension of y\_pred
155. **if** y.ndim != self.y\_pred.ndim:
156. **raise** TypeError(
157. 'y should have the same shape as self.y\_pred',
158. ('y', y.type, 'y\_pred', self.y\_pred.type)
159. )
160. # check if y is of the correct datatype
161. **if** y.dtype.startswith('int'):
162. # the T.neq operator returns a vector of 0s and 1s, where 1
163. # represents a mistake in prediction
164. **return** T.mean(T.neq(self.y\_pred, y))
165. **else**:
166. **raise** NotImplementedError()

169. **def** load\_data(dataset):
170. ''''' Loads the dataset
172. :type dataset: string
173. :param dataset: the path to the dataset (here MNIST)
174. '''
176. #############
177. # LOAD DATA #
178. #############
180. # Download the MNIST dataset if it is not present
181. data\_dir, data\_file = os.path.split(dataset)
182. **if** data\_dir == "" **and** **not** os.path.isfile(dataset):
183. # Check if dataset is in the data directory.
184. new\_path = os.path.join(
185. os.path.split(\_\_file\_\_)[0],
186. "..",
187. "data",
188. dataset
189. )
190. **if** os.path.isfile(new\_path) **or** data\_file == 'mnist.pkl.gz':
191. dataset = new\_path
193. **if** (**not** os.path.isfile(dataset)) **and** data\_file == 'mnist.pkl.gz':
194. **import** urllib
195. origin = (
196. 'http://www.iro.umontreal.ca/~lisa/deep/data/mnist/mnist.pkl.gz'
197. )
198. **print** 'Downloading data from %s' % origin
199. urllib.urlretrieve(origin, dataset)
201. **print** '... loading data'
203. # Load the dataset
204. f = gzip.open(dataset, 'rb')
205. train\_set, valid\_set, test\_set = cPickle.load(f)
206. f.close()
207. #train\_set, valid\_set, test\_set format: tuple(input, target)
208. #input is an numpy.ndarray of 2 dimensions (a matrix)
209. #witch row's correspond to an example. target is a
210. #numpy.ndarray of 1 dimensions (vector)) that have the same length as
211. #the number of rows in the input. It should give the target
212. #target to the example with the same index in the input.
214. **def** shared\_dataset(data\_xy, borrow=True):
215. """ Function that loads the dataset into shared variables
217. The reason we store our dataset in shared variables is to allow
218. Theano to copy it into the GPU memory (when code is run on GPU).
219. Since copying data into the GPU is slow, copying a minibatch everytime
220. is needed (the default behaviour if the data is not in a shared
221. variable) would lead to a large decrease in performance.
222. """
223. data\_x, data\_y = data\_xy
224. shared\_x = theano.shared(numpy.asarray(data\_x,
225. dtype=theano.config.floatX),
226. borrow=borrow)
227. shared\_y = theano.shared(numpy.asarray(data\_y,
228. dtype=theano.config.floatX),
229. borrow=borrow)
230. # When storing data on the GPU it has to be stored as floats
231. # therefore we will store the labels as ``floatX`` as well
232. # (``shared\_y`` does exactly that). But during our computations
233. # we need them as ints (we use labels as index, and if they are
234. # floats it doesn't make sense) therefore instead of returning
235. # ``shared\_y`` we will have to cast it to int. This little hack
236. # lets ous get around this issue
237. **return** shared\_x, T.cast(shared\_y, 'int32')
239. test\_set\_x, test\_set\_y = shared\_dataset(test\_set)
240. valid\_set\_x, valid\_set\_y = shared\_dataset(valid\_set)
241. train\_set\_x, train\_set\_y = shared\_dataset(train\_set)
243. rval = [(train\_set\_x, train\_set\_y), (valid\_set\_x, valid\_set\_y),
244. (test\_set\_x, test\_set\_y)]
245. **return** rval

248. **def** sgd\_optimization\_mnist(learning\_rate=0.13, n\_epochs=1000,
249. dataset='mnist.pkl.gz',
250. batch\_size=600):
251. """
252. Demonstrate stochastic gradient descent optimization of a log-linear
253. model
255. This is demonstrated on MNIST.
257. :type learning\_rate: float
258. :param learning\_rate: learning rate used (factor for the stochastic
259. gradient)
261. :type n\_epochs: int
262. :param n\_epochs: maximal number of epochs to run the optimizer
264. :type dataset: string
265. :param dataset: the path of the MNIST dataset file from
266. http://www.iro.umontreal.ca/~lisa/deep/data/mnist/mnist.pkl.gz
268. """
269. datasets = load\_data(dataset)
271. train\_set\_x, train\_set\_y = datasets[0]
272. valid\_set\_x, valid\_set\_y = datasets[1]
273. test\_set\_x, test\_set\_y = datasets[2]
275. # compute number of minibatches for training, validation and testing
276. n\_train\_batches = train\_set\_x.get\_value(borrow=True).shape[0] / batch\_size
277. n\_valid\_batches = valid\_set\_x.get\_value(borrow=True).shape[0] / batch\_size
278. n\_test\_batches = test\_set\_x.get\_value(borrow=True).shape[0] / batch\_size
280. ######################
281. # BUILD ACTUAL MODEL #
282. ######################
283. **print** '... building the model'
285. # allocate symbolic variables for the data
286. index = T.lscalar()  # index to a [mini]batch
288. # generate symbolic variables for input (x and y represent a
289. # minibatch)
290. x = T.matrix('x')  # data, presented as rasterized images
291. y = T.ivector('y')  # labels, presented as 1D vector of [int] labels
293. # construct the logistic regression class
294. # Each MNIST image has size 28\*28
295. classifier = LogisticRegression(input=x, n\_in=28 \* 28, n\_out=10)
297. # the cost we minimize during training is the negative log likelihood of
298. # the model in symbolic format
299. cost = classifier.negative\_log\_likelihood(y)
301. # compiling a Theano function that computes the mistakes that are made by
302. # the model on a minibatch
303. test\_model = theano.function(
304. inputs=[index],
305. outputs=classifier.errors(y),
306. givens={
307. x: test\_set\_x[index \* batch\_size: (index + 1) \* batch\_size],
308. y: test\_set\_y[index \* batch\_size: (index + 1) \* batch\_size]
309. }
310. )
312. validate\_model = theano.function(
313. inputs=[index],
314. outputs=classifier.errors(y),
315. givens={
316. x: valid\_set\_x[index \* batch\_size: (index + 1) \* batch\_size],
317. y: valid\_set\_y[index \* batch\_size: (index + 1) \* batch\_size]
318. }
319. )
321. # compute the gradient of cost with respect to theta = (W,b)
322. g\_W = T.grad(cost=cost, wrt=classifier.W)
323. g\_b = T.grad(cost=cost, wrt=classifier.b)
325. # start-snippet-3
326. # specify how to update the parameters of the model as a list of
327. # (variable, update expression) pairs.
328. updates = [(classifier.W, classifier.W - learning\_rate \* g\_W),
329. (classifier.b, classifier.b - learning\_rate \* g\_b)]
331. # compiling a Theano function `train\_model` that returns the cost, but in
332. # the same time updates the parameter of the model based on the rules
333. # defined in `updates`
334. train\_model = theano.function(
335. inputs=[index],
336. outputs=cost,
337. updates=updates,
338. givens={
339. x: train\_set\_x[index \* batch\_size: (index + 1) \* batch\_size],
340. y: train\_set\_y[index \* batch\_size: (index + 1) \* batch\_size]
341. }
342. )
343. # end-snippet-3
345. ###############
346. # TRAIN MODEL #
347. ###############
348. **print** '... training the model'
349. # early-stopping parameters
350. patience = 5000  # look as this many examples regardless
351. patience\_increase = 2  # wait this much longer when a new best is
352. # found
353. improvement\_threshold = 0.995  # a relative improvement of this much is
354. # considered significant
355. validation\_frequency = min(n\_train\_batches, patience / 2)
356. # go through this many
357. # minibatche before checking the network
358. # on the validation set; in this case we
359. # check every epoch
361. best\_validation\_loss = numpy.inf
362. test\_score = 0.
363. start\_time = time.clock()
365. done\_looping = False
366. epoch = 0
367. **while** (epoch < n\_epochs) **and** (**not** done\_looping):
368. epoch = epoch + 1
369. **for** minibatch\_index **in** xrange(n\_train\_batches):
371. minibatch\_avg\_cost = train\_model(minibatch\_index)
372. # iteration number
373. iter = (epoch - 1) \* n\_train\_batches + minibatch\_index
375. **if** (iter + 1) % validation\_frequency == 0:
376. # compute zero-one loss on validation set
377. validation\_losses = [validate\_model(i)
378. **for** i **in** xrange(n\_valid\_batches)]
379. this\_validation\_loss = numpy.mean(validation\_losses)
381. **print**(
382. 'epoch %i, minibatch %i/%i, validation error %f %%' %
383. (
384. epoch,
385. minibatch\_index + 1,
386. n\_train\_batches,
387. this\_validation\_loss \* 100.
388. )
389. )
391. # if we got the best validation score until now
392. **if** this\_validation\_loss < best\_validation\_loss:
393. #improve patience if loss improvement is good enough
394. **if** this\_validation\_loss < best\_validation\_loss \*  \
395. improvement\_threshold:
396. patience = max(patience, iter \* patience\_increase)
398. best\_validation\_loss = this\_validation\_loss
399. # test it on the test set
401. test\_losses = [test\_model(i)
402. **for** i **in** xrange(n\_test\_batches)]
403. test\_score = numpy.mean(test\_losses)
405. **print**(
406. (
407. '     epoch %i, minibatch %i/%i, test error of'
408. ' best model %f %%'
409. ) %
410. (
411. epoch,
412. minibatch\_index + 1,
413. n\_train\_batches,
414. test\_score \* 100.
415. )
416. )
418. **if** patience <= iter:
419. done\_looping = True
420. **break**
422. end\_time = time.clock()
423. **print**(
424. (
425. 'Optimization complete with best validation score of %f %%,'
426. 'with test performance %f %%'
427. )
428. % (best\_validation\_loss \* 100., test\_score \* 100.)
429. )
430. **print** 'The code run for %d epochs, with %f epochs/sec' % (
431. epoch, 1. \* epoch / (end\_time - start\_time))
432. **print** >> sys.stderr, ('The code for file ' +
433. os.path.split(\_\_file\_\_)[1] +
434. ' ran for %.1fs' % ((end\_time - start\_time)))
436. **if** \_\_name\_\_ == '\_\_main\_\_':
437. sgd\_optimization\_mnist()

用户可以学着使用SGD逻辑回归来分类MNIST数字，从dl教程文件夹中找到如下的代码执行：

1. python code/logistic\_sgd.py

输出会如下：

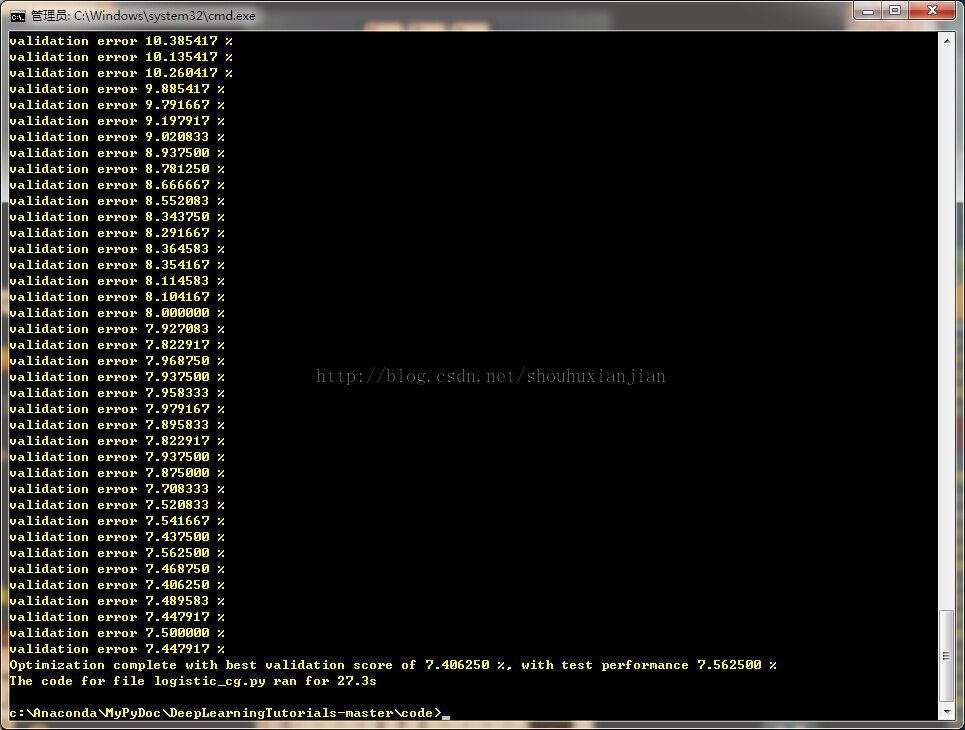
1. ...
2. epoch 72, minibatch 83/83, validation error 7.510417 %
3. epoch 72, minibatch 83/83, test error of best model 7.510417 %
4. epoch 73, minibatch 83/83, validation error 7.500000 %
5. epoch 73, minibatch 83/83, test error of best model 7.489583 %
6. Optimization complete with best validation score of 7.500000 %,with test performance 7.489583 %
7. The code run **for** 74 epochs, with 1.936983 epochs/sec

在intel（R）Core(TM)2 Duo CPU E8400 @ 3.00Ghz下，代码差不多速度为1.936 epochs/sec，在执行了75个epochs之后，达到的测试错误率为7.489%。在GPU上执行差不多是10.0epochs/sec。这个例子中我们的batch size为600.

**脚注：**

[1] 对于更小的数据集和更简单的模型，更多老练的下降方法将会更有效。例子代码[logistic\_cg.py](http://deeplearning.net/tutorial/code/logistic_cg.py)显示如何使用Scipy的共轭梯度 来在Theano上解决逻辑回归的问题。

下面是在win7\_64bit+cuda6.5+anaconda\_2.1.0+theano\_0.7.0下跑的logistic\_cg.py的结果：



**参考资料：**

[1] 官网：http://deeplearning.net/tutorial/logreg.html#logreg

[2] Classifying MNIST digits using Logistic Regression ：http://blog.sina.com.cn/s/blog\_6caa9fa10101m33n.html

[3]  DeepLearning tutorial（1）Softmax回归原理简介+代码详解 ：http://blog.csdn.net/u012162613/article/details/43157801

[4]

[**Theano3.4-练习之多层感知机**](http://www.cnblogs.com/shouhuxianjian/p/4564625.html)

来自http://deeplearning.net/tutorial/mlp.html#mlp

Multilayer Perceptron

note：这部分假设读者已经通读之前的一个练习 [*Classifying MNIST digits using Logistic Regression*](http://deeplearning.net/tutorial/logreg.html).（http://blog.csdn.net/shouhuxianjian/article/details/46375461）。另外，它使用新的theano函数和概念： [T.tanh](http://deeplearning.net/software/theano/tutorial/examples.html?highlight=tanh), [shared variables](http://deeplearning.net/software/theano/tutorial/examples.html#using-shared-variables), [basic arithmetic ops](http://deeplearning.net/software/theano/tutorial/adding.html#adding-two-scalars), [T.grad](http://deeplearning.net/software/theano/tutorial/examples.html#computing-gradients), [*L1 and L2 regularization*](http://deeplearning.net/tutorial/gettingstarted.html#l1-l2-regularization), [floatX](http://deeplearning.net/software/theano/library/config.html#config.floatX)。如果你想要在GPU上运行代码，记得看[GPU](http://deeplearning.net/software/theano/tutorial/using_gpu.html).

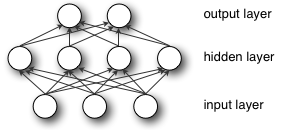
note：这部分的代码可以从这里下载[here](http://deeplearning.net/tutorial/code/mlp.py).

    接下来要呈现的使用theano的架构是单隐藏层多层感知机（MLP）。一个MLP可以被视为一个逻辑回归分类器，其中的输入首先通过学到的非线性 \Phi 来转换。该转换是将输入数据映射到一个空间中，在该空间中不同的类别可以线性可分。中间层也就是指隐藏层。一个隐藏层已经足够让MLPs成为一个通用的逼近器。然而我们随后看到的是在使用许多这样的隐藏层之后可以得到很大的好处，即深度学习的前提条件（指的是隐藏层必须超过一层）。可以看这些课程的笔记：[ntroduction to MLPs, the back-propagation algorithm, and how to train MLPs](http://www.iro.umontreal.ca/~pift6266/H10/notes/mlp.html).

该教程依然是在MNIST数字分类上来介绍的。

**一、模型**

    有着单一隐藏层的MLP（或者人工神经网络，ANN）的图示如下：



    正式的说，一层隐藏层MLP表示为函数形式： f: R^D \rightarrowR^L，这里D是输入向量 x的size， L是输出向量f(x)的size，表示矩阵符号形式如下：

f(x) = G( b^{(2)} + W^{(2)}( s( b^{(1)} + W^{(1)} x))),

有着偏置向量 b^{(1)}, b^{(2)}，权重矩阵 W^{(1)}, W^{(2)}，激活函数G 和s。

    向量h(x) = \Phi(x) = s(b^{(1)} + W^{(1)} x)构成这个隐藏层。W^{(1)} \in R^{D \times D_h}是连接输入向量到隐藏层之间的权重矩阵。每一列W^{(1)}_{\cdot i} 表示输入单元到第 i 个隐藏单元的权重。对s 的选择通常是tanh： tanh(a)=(e^a-e^{-a})/(e^a+e^{-a})或者逻辑sigmoid函数： sigmoid(a)=1/(1+e^{-a})。我们在这个教程中将会使用tanh，因为它的训练速度一般可以更快（而且有时候有着更好的局部最小）。tanh和sigmoid都是标量to标量的函数，不过它们自然的扩展到向量和张量的时候都是逐元素计算的（例如，在向量的每个元素上独立计算，生成一个同样size的向量）。

    输出向量计算结果为：o(x) = G(b^{(2)} + W^{(2)} h(x))。读者应该在前一个练习（Theano3.3-练习之逻辑回归）就该看过这个形式了，和之前一样，属于哪一类的概率可以通过选择G 为softmax函数来计算得到（多类分类情况下）。

    为了训练一个MLP，我们需要对这个模型的所有参数进行学习，这里我们使用带有minibatch的 [*Stochastic Gradient Descent*](http://deeplearning.net/tutorial/gettingstarted.html#opt-sgd)*。*需要学习的参数集就是： \theta =\{W^{(2)},b^{(2)},W^{(1)},b^{(1)}\}。可以通过BP算法（导数链式规则的特殊情况）来得到梯度 \partial{\ell}/\partial{\theta}。不过幸运的是，因为theano可以自动的进行求导微分，我们不需要在本教程中介绍如何求导

**二、从LR到MLP**

    该教程关注的是单隐藏层的MLP。所以先编写单层隐藏层的类。为了构造这个MLP，我们随后只需要在顶部放上一个逻辑回归层就好：

1. **class** HiddenLayer(object):
2. **def** \_\_init\_\_(self, rng, input, n\_in, n\_out, W=None, b=None,
3. activation=T.tanh):
4. """
5. Typical hidden layer of a MLP: units are fully-connected and have
6. sigmoidal activation function. Weight matrix W is of shape (n\_in,n\_out)
7. and the bias vector b is of shape (n\_out,).
9. NOTE : The nonlinearity used here is tanh
11. Hidden unit activation is given by: tanh(dot(input,W) + b)
13. :type rng: numpy.random.RandomState
14. :param rng: a random number generator used to initialize weights
16. :type input: theano.tensor.dmatrix
17. :param input: a symbolic tensor of shape (n\_examples, n\_in)
19. :type n\_in: int
20. :param n\_in: dimensionality of input
22. :type n\_out: int
23. :param n\_out: number of hidden units
25. :type activation: theano.Op or function
26. :param activation: Non linearity to be applied in the hidden
27. layer
28. """
29. self.input = input

隐藏层 i 的权重的初始化值需要从依赖于激活函数的对称间隔上统一采样得到。对于tanh激活函数，在 [[Xavier10]](http://deeplearning.net/tutorial/references.html#xavier10) 中的获得的结果上来看，这个间隔应该是[-\sqrt{\frac{6}{fan_{in}+fan_{out}}},\sqrt{\frac{6}{fan_{in}+fan_{out}}}]。这里fan_{in} 是第 (i-1)-th层的单元个数， fan_{out}是第i-th层的单元个数。对于sigmoid函数来说，间隔是 [-4\sqrt{\frac{6}{fan_{in}+fan_{out}}},4\sqrt{\frac{6}{fan_{in}+fan_{out}}}]。在训练的早期，这个初始化是可以确保每个神经元会在它的激活函数的变化较大的区域部分，使得能够很容易往上传播（从输入到输出方向）和往回传播（梯度从输出到输入方向）：

1. # `W` is initialized with `W\_values` which is uniformely sampled
2. # from sqrt(-6./(n\_in+n\_hidden)) and sqrt(6./(n\_in+n\_hidden))
3. # for tanh activation function
4. # the output of uniform if converted using asarray to dtype
5. # theano.config.floatX so that the code is runable on GPU
6. # Note : optimal initialization of weights is dependent on the
7. #        activation function used (among other things).
8. #        For example, results presented in [Xavier10] suggest that you
9. #        should use 4 times larger initial weights for sigmoid
10. #        compared to tanh
11. #        We have no info for other function, so we use the same as
12. #        tanh.
13. **if** W **is** None:
14. W\_values = numpy.asarray(
15. rng.uniform(
16. low=-numpy.sqrt(6. / (n\_in + n\_out)),
17. high=numpy.sqrt(6. / (n\_in + n\_out)),
18. size=(n\_in, n\_out)
19. ),
20. dtype=theano.config.floatX
21. )
22. **if** activation == theano.tensor.nnet.sigmoid:
23. W\_values \*= 4
25. W = theano.shared(value=W\_values, name='W', borrow=True)
27. **if** b **is** None:
28. b\_values = numpy.zeros((n\_out,), dtype=theano.config.floatX)
29. b = theano.shared(value=b\_values, name='b', borrow=True)
31. self.W = W
32. self.b = b

注意到我们使用了一个给定的非线性函数作为隐藏层的激活函数。默认情况下是tanh，不过在许多情况下我们想使用其他激活函数：

1. lin\_output = T.dot(input, self.W) + self.b
2. self.output = (
3. lin\_output **if** activation **is** None
4. **else** activation(lin\_output)
5. )

    如果深入原理部分，这个类实现graph的时候需要计算隐藏层的值 h(x) = \Phi(x) = s(b^{(1)} + W^{(1)} x)。如果给graph的输入和LogisticRegression类一样，就像之前的教程一样，就可以得到MLP的输出。下面的是MLP类的简单实现代码：

1. **class** MLP(object):
2. """Multi-Layer Perceptron Class
4. A multilayer perceptron is a feedforward artificial neural network model
5. that has one layer or more of hidden units and nonlinear activations.
6. Intermediate layers usually have as activation function tanh or the
7. sigmoid function (defined here by a ``HiddenLayer`` class)  while the
8. top layer is a softmax layer (defined here by a ``LogisticRegression``
9. class).
10. """
12. **def** \_\_init\_\_(self, rng, input, n\_in, n\_hidden, n\_out):
13. """Initialize the parameters for the multilayer perceptron
15. :type rng: numpy.random.RandomState
16. :param rng: a random number generator used to initialize weights
18. :type input: theano.tensor.TensorType
19. :param input: symbolic variable that describes the input of the
20. architecture (one minibatch)
22. :type n\_in: int
23. :param n\_in: number of input units, the dimension of the space in
24. which the datapoints lie
26. :type n\_hidden: int
27. :param n\_hidden: number of hidden units
29. :type n\_out: int
30. :param n\_out: number of output units, the dimension of the space in
31. which the labels lie
33. """
35. # Since we are dealing with a one hidden layer MLP, this will translate
36. # into a HiddenLayer with a tanh activation function connected to the
37. # LogisticRegression layer; the activation function can be replaced by
38. # sigmoid or any other nonlinear function
39. self.hiddenLayer = HiddenLayer(
40. rng=rng,
41. input=input,
42. n\_in=n\_in,
43. n\_out=n\_hidden,
44. activation=T.tanh
45. )
47. # The logistic regression layer gets as input the hidden units
48. # of the hidden layer
49. self.logRegressionLayer = LogisticRegression(
50. input=self.hiddenLayer.output,
51. n\_in=n\_hidden,
52. n\_out=n\_out
53. )

    在这个教程中，我们同样会使用L1和L2正则化（ [*L1 and L2 regularization*](http://deeplearning.net/tutorial/gettingstarted.html#l1-l2-regularization)）。同时我们需要计算L1范数和权重 W^{(1)}, W^{(2)}的L2范数的平方：

1. # L1 norm ; one regularization option is to enforce L1 norm to
2. # be small
3. self.L1 = (
4. abs(self.hiddenLayer.W).sum()
5. + abs(self.logRegressionLayer.W).sum()
6. )
8. # square of L2 norm ; one regularization option is to enforce
9. # square of L2 norm to be small
10. self.L2\_sqr = (
11. (self.hiddenLayer.W \*\* 2).sum()
12. + (self.logRegressionLayer.W \*\* 2).sum()
13. )
15. # negative log likelihood of the MLP is given by the negative
16. # log likelihood of the output of the model, computed in the
17. # logistic regression layer
18. self.negative\_log\_likelihood = (
19. self.logRegressionLayer.negative\_log\_likelihood
20. )
21. # same holds for the function computing the number of errors
22. self.errors = self.logRegressionLayer.errors
24. # the parameters of the model are the parameters of the two layer it is
25. # made out of
26. self.params = self.hiddenLayer.params + self.logRegressionLayer.params

    就像之前一样，通过MSGD来训练，不同之处在于我们会修改cost函数，使得它包含正则化项。L1\_reg和L2\_reg是用来控制整个cost函数中的正则化项权重的超参数。新的cost的代码如下：

1. # the cost we minimize during training is the negative log likelihood of
2. # the model plus the regularization terms (L1 and L2); cost is expressed
3. # here symbolically
4. cost = (
5. classifier.negative\_log\_likelihood(y)
6. + L1\_reg \* classifier.L1
7. + L2\_reg \* classifier.L2\_sqr
8. )

    然后使用梯度来更新模型的参数。这里的代码差不多和逻辑回归的代码一样。只有参数的个数不同。为了避开这个问题（然代码可以用在任意数量的参数上），我们将会创建带有params的模型来生成参数列表然后对它进行解析，每一步计算一个替代：

1. # compute the gradient of cost with respect to theta (sotred in params)
2. # the resulting gradients will be stored in a list gparams
3. gparams = [T.grad(cost, param) **for** param **in** classifier.params]
5. # specify how to update the parameters of the model as a list of
6. # (variable, update expression) pairs
8. # given two list the zip A = [a1, a2, a3, a4] and B = [b1, b2, b3, b4] of
9. # same length, zip generates a list C of same size, where each element
10. # is a pair formed from the two lists :
11. #    C = [(a1, b1), (a2, b2), (a3, b3), (a4, b4)]
12. updates = [
13. (param, param - learning\_rate \* gparam)
14. **for** param, gparam **in** zip(classifier.params, gparams)
15. ]
17. # compiling a Theano function `train\_model` that returns the cost, but
18. # in the same time updates the parameter of the model based on the rules
19. # defined in `updates`
20. train\_model = theano.function(
21. inputs=[index],
22. outputs=cost,
23. updates=updates,
24. givens={
25. x: train\_set\_x[index \* batch\_size: (index + 1) \* batch\_size],
26. y: train\_set\_y[index \* batch\_size: (index + 1) \* batch\_size]
27. }
28. )

**三、将上面的部分合并到一起**

    在了解了基本的概念之后，写一个MLP类变得非常容易。下面的代码就是过程，类似于之前的LR实现的方式：

1. """
2. This tutorial introduces the multilayer perceptron using Theano.
4. A multilayer perceptron is a logistic regressor where
5. instead of feeding the input to the logistic regression you insert a
6. intermediate layer, called the hidden layer, that has a nonlinear
7. activation function (usually tanh or sigmoid) . One can use many such
8. hidden layers making the architecture deep. The tutorial will also tackle
9. the problem of MNIST digit classification.
11. .. math::
13. f(x) = G( b^{(2)} + W^{(2)}( s( b^{(1)} + W^{(1)} x))),
15. References:
17. - textbooks: "Pattern Recognition and Machine Learning" -
18. Christopher M. Bishop, section 5
20. """
21. \_\_docformat\_\_ = 'restructedtext en'

24. **import** os
25. **import** sys
26. **import** time
28. **import** numpy
30. **import** theano
31. **import** theano.tensor as T

34. **from** logistic\_sgd **import** LogisticRegression, load\_data

37. # start-snippet-1
38. **class** HiddenLayer(object):
39. **def** \_\_init\_\_(self, rng, input, n\_in, n\_out, W=None, b=None,
40. activation=T.tanh):
41. """
42. Typical hidden layer of a MLP: units are fully-connected and have
43. sigmoidal activation function. Weight matrix W is of shape (n\_in,n\_out)
44. and the bias vector b is of shape (n\_out,).
46. NOTE : The nonlinearity used here is tanh
48. Hidden unit activation is given by: tanh(dot(input,W) + b)
50. :type rng: numpy.random.RandomState
51. :param rng: a random number generator used to initialize weights
53. :type input: theano.tensor.dmatrix
54. :param input: a symbolic tensor of shape (n\_examples, n\_in)
56. :type n\_in: int
57. :param n\_in: dimensionality of input
59. :type n\_out: int
60. :param n\_out: number of hidden units
62. :type activation: theano.Op or function
63. :param activation: Non linearity to be applied in the hidden
64. layer
65. """
66. self.input = input
67. # end-snippet-1
69. # `W` is initialized with `W\_values` which is uniformely sampled
70. # from sqrt(-6./(n\_in+n\_hidden)) and sqrt(6./(n\_in+n\_hidden))
71. # for tanh activation function
72. # the output of uniform if converted using asarray to dtype
73. # theano.config.floatX so that the code is runable on GPU
74. # Note : optimal initialization of weights is dependent on the
75. #        activation function used (among other things).
76. #        For example, results presented in [Xavier10] suggest that you
77. #        should use 4 times larger initial weights for sigmoid
78. #        compared to tanh
79. #        We have no info for other function, so we use the same as
80. #        tanh.
81. **if** W **is** None:
82. W\_values = numpy.asarray(
83. rng.uniform(
84. low=-numpy.sqrt(6. / (n\_in + n\_out)),
85. high=numpy.sqrt(6. / (n\_in + n\_out)),
86. size=(n\_in, n\_out)
87. ),
88. dtype=theano.config.floatX
89. )
90. **if** activation == theano.tensor.nnet.sigmoid:
91. W\_values \*= 4
93. W = theano.shared(value=W\_values, name='W', borrow=True)
95. **if** b **is** None:
96. b\_values = numpy.zeros((n\_out,), dtype=theano.config.floatX)
97. b = theano.shared(value=b\_values, name='b', borrow=True)
99. self.W = W
100. self.b = b
102. lin\_output = T.dot(input, self.W) + self.b
103. self.output = (
104. lin\_output **if** activation **is** None
105. **else** activation(lin\_output)
106. )
107. # parameters of the model
108. self.params = [self.W, self.b]

111. # start-snippet-2
112. **class** MLP(object):
113. """Multi-Layer Perceptron Class
115. A multilayer perceptron is a feedforward artificial neural network model
116. that has one layer or more of hidden units and nonlinear activations.
117. Intermediate layers usually have as activation function tanh or the
118. sigmoid function (defined here by a ``HiddenLayer`` class)  while the
119. top layer is a softmax layer (defined here by a ``LogisticRegression``
120. class).
121. """
123. **def** \_\_init\_\_(self, rng, input, n\_in, n\_hidden, n\_out):
124. """Initialize the parameters for the multilayer perceptron
126. :type rng: numpy.random.RandomState
127. :param rng: a random number generator used to initialize weights
129. :type input: theano.tensor.TensorType
130. :param input: symbolic variable that describes the input of the
131. architecture (one minibatch)
133. :type n\_in: int
134. :param n\_in: number of input units, the dimension of the space in
135. which the datapoints lie
137. :type n\_hidden: int
138. :param n\_hidden: number of hidden units
140. :type n\_out: int
141. :param n\_out: number of output units, the dimension of the space in
142. which the labels lie
144. """
146. # Since we are dealing with a one hidden layer MLP, this will translate
147. # into a HiddenLayer with a tanh activation function connected to the
148. # LogisticRegression layer; the activation function can be replaced by
149. # sigmoid or any other nonlinear function
150. self.hiddenLayer = HiddenLayer(
151. rng=rng,
152. input=input,
153. n\_in=n\_in,
154. n\_out=n\_hidden,
155. activation=T.tanh
156. )
158. # The logistic regression layer gets as input the hidden units
159. # of the hidden layer
160. self.logRegressionLayer = LogisticRegression(
161. input=self.hiddenLayer.output,
162. n\_in=n\_hidden,
163. n\_out=n\_out
164. )
165. # end-snippet-2 start-snippet-3
166. # L1 norm ; one regularization option is to enforce L1 norm to
167. # be small
168. self.L1 = (
169. abs(self.hiddenLayer.W).sum()
170. + abs(self.logRegressionLayer.W).sum()
171. )
173. # square of L2 norm ; one regularization option is to enforce
174. # square of L2 norm to be small
175. self.L2\_sqr = (
176. (self.hiddenLayer.W \*\* 2).sum()
177. + (self.logRegressionLayer.W \*\* 2).sum()
178. )
180. # negative log likelihood of the MLP is given by the negative
181. # log likelihood of the output of the model, computed in the
182. # logistic regression layer
183. self.negative\_log\_likelihood = (
184. self.logRegressionLayer.negative\_log\_likelihood
185. )
186. # same holds for the function computing the number of errors
187. self.errors = self.logRegressionLayer.errors
189. # the parameters of the model are the parameters of the two layer it is
190. # made out of
191. self.params = self.hiddenLayer.params + self.logRegressionLayer.params
192. # end-snippet-3

195. **def** test\_mlp(learning\_rate=0.01, L1\_reg=0.00, L2\_reg=0.0001, n\_epochs=1000,
196. dataset='mnist.pkl.gz', batch\_size=20, n\_hidden=500):
197. """
198. Demonstrate stochastic gradient descent optimization for a multilayer
199. perceptron
201. This is demonstrated on MNIST.
203. :type learning\_rate: float
204. :param learning\_rate: learning rate used (factor for the stochastic
205. gradient
207. :type L1\_reg: float
208. :param L1\_reg: L1-norm's weight when added to the cost (see
209. regularization)
211. :type L2\_reg: float
212. :param L2\_reg: L2-norm's weight when added to the cost (see
213. regularization)
215. :type n\_epochs: int
216. :param n\_epochs: maximal number of epochs to run the optimizer
218. :type dataset: string
219. :param dataset: the path of the MNIST dataset file from
220. http://www.iro.umontreal.ca/~lisa/deep/data/mnist/mnist.pkl.gz

223. """
224. datasets = load\_data(dataset)
226. train\_set\_x, train\_set\_y = datasets[0]
227. valid\_set\_x, valid\_set\_y = datasets[1]
228. test\_set\_x, test\_set\_y = datasets[2]
230. # compute number of minibatches for training, validation and testing
231. n\_train\_batches = train\_set\_x.get\_value(borrow=True).shape[0] / batch\_size
232. n\_valid\_batches = valid\_set\_x.get\_value(borrow=True).shape[0] / batch\_size
233. n\_test\_batches = test\_set\_x.get\_value(borrow=True).shape[0] / batch\_size
235. ######################
236. # BUILD ACTUAL MODEL #
237. ######################
238. **print** '... building the model'
240. # allocate symbolic variables for the data
241. index = T.lscalar()  # index to a [mini]batch
242. x = T.matrix('x')  # the data is presented as rasterized images
243. y = T.ivector('y')  # the labels are presented as 1D vector of
244. # [int] labels
246. rng = numpy.random.RandomState(1234)
248. # construct the MLP class
249. classifier = MLP(
250. rng=rng,
251. input=x,
252. n\_in=28 \* 28,
253. n\_hidden=n\_hidden,
254. n\_out=10
255. )
257. # start-snippet-4
258. # the cost we minimize during training is the negative log likelihood of
259. # the model plus the regularization terms (L1 and L2); cost is expressed
260. # here symbolically
261. cost = (
262. classifier.negative\_log\_likelihood(y)
263. + L1\_reg \* classifier.L1
264. + L2\_reg \* classifier.L2\_sqr
265. )
266. # end-snippet-4
268. # compiling a Theano function that computes the mistakes that are made
269. # by the model on a minibatch
270. test\_model = theano.function(
271. inputs=[index],
272. outputs=classifier.errors(y),
273. givens={
274. x: test\_set\_x[index \* batch\_size:(index + 1) \* batch\_size],
275. y: test\_set\_y[index \* batch\_size:(index + 1) \* batch\_size]
276. }
277. )
279. validate\_model = theano.function(
280. inputs=[index],
281. outputs=classifier.errors(y),
282. givens={
283. x: valid\_set\_x[index \* batch\_size:(index + 1) \* batch\_size],
284. y: valid\_set\_y[index \* batch\_size:(index + 1) \* batch\_size]
285. }
286. )
288. # start-snippet-5
289. # compute the gradient of cost with respect to theta (sotred in params)
290. # the resulting gradients will be stored in a list gparams
291. gparams = [T.grad(cost, param) **for** param **in** classifier.params]
293. # specify how to update the parameters of the model as a list of
294. # (variable, update expression) pairs
296. # given two list the zip A = [a1, a2, a3, a4] and B = [b1, b2, b3, b4] of
297. # same length, zip generates a list C of same size, where each element
298. # is a pair formed from the two lists :
299. #    C = [(a1, b1), (a2, b2), (a3, b3), (a4, b4)]
300. updates = [
301. (param, param - learning\_rate \* gparam)
302. **for** param, gparam **in** zip(classifier.params, gparams)
303. ]
305. # compiling a Theano function `train\_model` that returns the cost, but
306. # in the same time updates the parameter of the model based on the rules
307. # defined in `updates`
308. train\_model = theano.function(
309. inputs=[index],
310. outputs=cost,
311. updates=updates,
312. givens={
313. x: train\_set\_x[index \* batch\_size: (index + 1) \* batch\_size],
314. y: train\_set\_y[index \* batch\_size: (index + 1) \* batch\_size]
315. }
316. )
317. # end-snippet-5
319. ###############
320. # TRAIN MODEL #
321. ###############
322. **print** '... training'
324. # early-stopping parameters
325. patience = 10000  # look as this many examples regardless
326. patience\_increase = 2  # wait this much longer when a new best is
327. # found
328. improvement\_threshold = 0.995  # a relative improvement of this much is
329. # considered significant
330. validation\_frequency = min(n\_train\_batches, patience / 2)
331. # go through this many
332. # minibatche before checking the network
333. # on the validation set; in this case we
334. # check every epoch
336. best\_validation\_loss = numpy.inf
337. best\_iter = 0
338. test\_score = 0.
339. start\_time = time.clock()
341. epoch = 0
342. done\_looping = False
344. **while** (epoch < n\_epochs) **and** (**not** done\_looping):
345. epoch = epoch + 1
346. **for** minibatch\_index **in** xrange(n\_train\_batches):
348. minibatch\_avg\_cost = train\_model(minibatch\_index)
349. # iteration number
350. iter = (epoch - 1) \* n\_train\_batches + minibatch\_index
352. **if** (iter + 1) % validation\_frequency == 0:
353. # compute zero-one loss on validation set
354. validation\_losses = [validate\_model(i) **for** i
355. **in** xrange(n\_valid\_batches)]
356. this\_validation\_loss = numpy.mean(validation\_losses)
358. **print**(
359. 'epoch %i, minibatch %i/%i, validation error %f %%' %
360. (
361. epoch,
362. minibatch\_index + 1,
363. n\_train\_batches,
364. this\_validation\_loss \* 100.
365. )
366. )
368. # if we got the best validation score until now
369. **if** this\_validation\_loss < best\_validation\_loss:
370. #improve patience if loss improvement is good enough
371. **if** (
372. this\_validation\_loss < best\_validation\_loss \*
373. improvement\_threshold
374. ):
375. patience = max(patience, iter \* patience\_increase)
377. best\_validation\_loss = this\_validation\_loss
378. best\_iter = iter
380. # test it on the test set
381. test\_losses = [test\_model(i) **for** i
382. **in** xrange(n\_test\_batches)]
383. test\_score = numpy.mean(test\_losses)
385. **print**(('     epoch %i, minibatch %i/%i, test error of '
386. 'best model %f %%') %
387. (epoch, minibatch\_index + 1, n\_train\_batches,
388. test\_score \* 100.))
390. **if** patience <= iter:
391. done\_looping = True
392. **break**
394. end\_time = time.clock()
395. **print**(('Optimization complete. Best validation score of %f %% '
396. 'obtained at iteration %i, with test performance %f %%') %
397. (best\_validation\_loss \* 100., best\_iter + 1, test\_score \* 100.))
398. **print** >> sys.stderr, ('The code for file ' +
399. os.path.split(\_\_file\_\_)[1] +
400. ' ran for %.2fm' % ((end\_time - start\_time) / 60.))

403. **if** \_\_name\_\_ == '\_\_main\_\_':
404. test\_mlp()

    用户可以如下这样运行这个代码：

python code/mlp.py

    输出会有如下的形式：

Optimization complete. Best validation score of 1.690000 % obtained at iteration 2070000, with test performance 1.650000 %

The code **for** file mlp.py ran **for** 97.34m

    在Intel(R) Core(TM) i7-2600K CPU @ 3.40GHz 上，该代码的速度大约为10.3 epoch/minute，并且在828 epochs的时候达到了测试错误率为1.65%。为了更好的了解MNIST上的结果，推荐读者去 [this](http://yann.lecun.com/exdb/mnist) 看看不同算法结果比较。

**四、训练MLPs的提示和技巧**

    在上面的代码中有许多超参数，它们不是被（通常来说也不能被）梯度下降而优化的。严格来说，找到一组最优超参数的值不是个容易解决的问题。首先，我们不能简单的将它们独立的进行优化。其次，我们不能容易的使用和前面介绍的梯度技术来处理（部分原因是因为一些参数离散值而另一些是实值）。第三，这个最优化问题不是凸优化和找到一个（局部）最小值的工作量可不小。好消息是在过去的25年中，研究者发明了各种经验规则来选择NN中的超参数。一个非常好的有关这些技巧的综述是由Yann LeCun ,Leon Bottou, Genevieve Orr, and Klaus-Robert Mueller写的 [Efficient BackProp](http://yann.lecun.com/exdb/publis/pdf/lecun-98b.pdf)。这里，我们归纳下这些同样的问题，并重点关注我们代码中实际用到的参数和技术。

**非线性**

    两个最常用的激活函数就是tanh和sigmoid函数。和 [Section 4.4](http://yann.lecun.com/exdb/publis/pdf/lecun-98b.pdf),中解释的原因一样，这两个非线性是中心对称的，这它们就能在下一层的时候生成的是0均值的输入（这是一个理想的属性）。经验上来时，我们发现tanh有着更好的收敛特性。（当然在2015年现在有relu和prelu等其他的激活函数，有兴趣的可以了解下）。

**权重初始化**

    在初始化的时候，我们想要权重围绕着原点（即数值0）足够小，这样激活函数就能呈现线性操作的趋势（这个看了sigmoid的函数图就能明白，在0点附近趋近于线性），在这个区域上梯度是最大的。其他理想的特性，特别对于深度网络来说，保存的激活函数的方差就像是从层到层的BP梯度的方差一样。这使得信息能够在网络中向上和向下传播，并且减少层间的差异。在某些假设的基础上，一个介于这两个约束条件的折中会导致有下面的初始化区别：

                                            tanh的初始化：uniform[-\frac{\sqrt{6}}{\sqrt{fan_{in}+fan_{out}}},\frac{\sqrt{6}}{\sqrt{fan_{in}+fan_{out}}}]

                                     sigmoid的初始化：uniform[-4*\frac{\sqrt{6}}{\sqrt{fan_{in}+fan_{out}}},4*\frac{\sqrt{6}}{\sqrt{fan_{in}+fan_{out}}}]

这里 fan_{in}是输入的个数， fan_{out}是隐藏单元的个数。数学上的思考可以参考 [[Xavier10]](http://deeplearning.net/tutorial/references.html#xavier10)。

**学习率**

    有许多文献是关注于如何选取一个好的学习率。最简单的解决方法就是简单的选择一个常量。经验规则：尝试几个log空间的值（10^{-1},10^{-2},\ldots），并缩小（对数）网格区域搜索到你得到的最低验证集误差的那个区域。

    随着时间来降低学习率是一个好想法，简单的方法就是 \frac{\mu_0}{1 + d\times t}，这里 \mu_0是初始化率（一般是用上面说的网格搜索技术来选择的）， d被称为“下降常量”用来控制学习率下降的速率（通常来说，是一个更小的正数，10^{-3} 或者更小）， t 是epoch//stage。

[Section 4.7](http://yann.lecun.com/exdb/publis/pdf/lecun-98b.pdf) 详细介绍了为每个参数（权重）选择一个学习率的过程,和基于分类器的误差来自适应的对它们进行选择。

**隐藏单元个数**

    超参数是非常的数据集依赖的。含糊的说，更复杂的输入分布就需要具有更大能力（capacity）的网络来对它进行建模，同样的也就需要更多的隐藏单元（注意到一层中权重的个数，这通常是一个更加直观的可以用来测量网络能力（capacity）的方法，也就是 D\times D_h(D是输入单元的数量，而D_h是隐藏单元的数量)）。

    除非我们使用一些正则化方案（早期停止或者L1/L2惩罚），隐藏单元个数 vs 泛化效果graph这两者呈现的是U的形状（即在中间某个点上是最好的权衡点，两头都是独立上升的）。

**正则化参数**

    通常用来试探L1/L2正则化参数 \lambda 的值是 10^{-2},10^{-3},\ldots。在这个框架中，我们到目前介绍的 优化这些参数不会明显的得到更好的结果，不过却值得探索。

**参考资料：**

[1] 官网：http://deeplearning.net/tutorial/mlp.html#mlp

[2] Deep learning with Theano 官方中文教程（翻译）（三）——多层感知机（MLP）：http://www.cnblogs.com/charleshuang/p/3648804.html

[**Theano3.5-练习之深度卷积网络**](http://www.cnblogs.com/shouhuxianjian/p/4564624.html)

来源：http://deeplearning.net/tutorial/lenet.html#lenet

Convolutional Neural Networks (LeNet)

note：这部分假设读者已经看过（[Theano3.3-练习之逻辑回归](http://blog.csdn.net/shouhuxianjian/article/details/46375461)）和（[Theano3.4-练习之多层感知机](http://blog.csdn.net/shouhuxianjian/article/details/46400947)）。另外，这里是用新的theano函数和概念： [T.tanh](http://deeplearning.net/software/theano/tutorial/examples.html?highlight=tanh),  [shared variables](http://deeplearning.net/software/theano/tutorial/examples.html#using-shared-variables),  [basic arithmetic ops](http://deeplearning.net/software/theano/tutorial/adding.html#adding-two-scalars), [T.grad](http://deeplearning.net/software/theano/tutorial/examples.html#computing-gradients), [floatX](http://deeplearning.net/software/theano/library/config.html#config.floatX),[downsample](http://deeplearning.net/software/theano/library/tensor/signal/downsample.html) , [conv2d](http://deeplearning.net/software/theano/library/tensor/signal/conv.html#module-conv), [dimshuffle](http://deeplearning.net/software/theano/library/tensor/basic.html#tensor._tensor_py_operators.dimshuffle).如果你想要在GPU上跑，记得看看 [GPU](http://deeplearning.net/software/theano/tutorial/using_gpu.html)。为了在GPU上跑这个例子，你需要一个好的GPU，至少需要1GB的显存。如果你的显示器也是连接着这个GPU，那么就需要注意一些事情了。因为GPU连接着显示器的时候，在每个GPU的函数调用的时候都会有几秒的限制。这是因为在当显示器需要请求GPU的时候，是无法进行计算的。没有这个限制的时候，你的屏幕看起来就像是死机一样，而且还会保持很长时间。（该例子就是在中端GPU下遇到的这个问题，这句话是原作者遇到的）当GPU没有连接到显示器的时候，就没有时间限制了，你可以降低batch size来解决这个时间延迟的问题。

note：这部分的可用代码可以从这里 [here](http://deeplearning.net/tutorial/code/convolutional_mlp.py)和这里 [3wolfmoon image](https://raw.githubusercontent.com/lisa-lab/DeepLearningTutorials/master/doc/images/3wolfmoon.jpg)下到。

**一、动机**

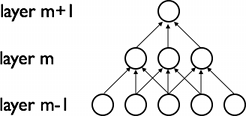
    卷积神经网络（Convolutional Neural Networks ，CNN）是MLPs的生物启发的变种。从Hubel和Wiesel的早期在猫的视觉皮层上的工作[[Hubel68]](http://deeplearning.net/tutorial/references.html#hubel68)上来看，我们知道视觉皮层包含着许多复杂排列的细胞。这些细胞对于视觉区域中小的子区域是非常敏感的，叫做感受野。这些子区域可以平铺从而覆盖整个视觉区域。这些细胞扮演着基于输入空间的局部过滤器而且很适合用来探索自然图像中的空间局部的强相关性。

    另外，两类基本的细胞类型已经被发现：简单细胞当检测到它们感受野的类边缘模式的时候的响应是最大的。复杂的细胞有着更大的感受野，而且对于陌生的提取的位置具有局部不变性。

    动物的视觉皮层是现今最好的视觉处理系统，所以我们很自然的去效仿它的原理。因此，能够在文献中找到许多神经启发的模型。比如：NeoCogitron  [[Fukushima]](http://deeplearning.net/tutorial/references.html#fukushima), HMAX [[Serre07]](http://deeplearning.net/tutorial/references.html#serre07)and LeNet-5 [[LeCun98]](http://deeplearning.net/tutorial/references.html#lecun98),本教程关注的是第三个模型。

**二、稀疏链接**

    CNN是通过在毗连的层的神经元之间建立局部连接模式来达到空间局部化的相关性。换句话说，第 m 层的隐藏单元的输入就是来自于第 m -1 层单元的子集，这些单元有着空间连续感受野。我们可以以下图示意：

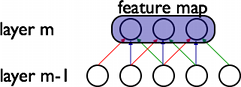


    想象下第m -1 层是视网膜作为输入。在上面部分，第 m 层的单元在基于视网膜输入的基础上有着width 为3的感受野，所以在视网膜层就只有3个毗连的神经元与上层的一个单元相连。第 m +1 层的单元与下层有着相似的连接，这里关于下层的感受野也同样是3，不过它们关于视网膜输入层的感受野是5（比3大）。每个单元对于视网膜上感受野外部区域的响应是无变化的（没反应也就是）。这个结构因此就能确保学到的“过滤器”能够对空间局部输入模式生成最强的响应。

    然而，正如上面介绍的，堆叠许多这样的（非线性）“过滤器”层就能够增加“全局”性（即，能够对更大的像素空间进行响应）。例如，第m+1的隐藏层中的单元能够编码一个有着width为5（在像素空间中的单位）的非线性特征。

**三、共享权重**

    另外，在CNN中，每个过滤器 h_i在整个视觉区域上是交叉重复的。在一个特征图中这些重复的单元共享相同的参数（权重向量和偏置）：



    在上面的图中，我们展示的是属于同一个特征图的3个隐藏单元。具有相同颜色的权重的值是相同的。梯度下降仍然可以用来学习这样的共享的参数，只是需要稍微改动下原来的算法。共享权重的梯度是简单的共享的参数的梯度的和。

    以这种形式来重复的单元允许检测到的特征在视觉区域中无视它们所处的位置。另外权重共享通过大量的减少了所需要学习的自由参数的数量而提升了学习的效率。这个模型上的约束条件能够保证CNN在视觉问题上得到更好的泛化。

**四、细节和符号介绍**

    一个特征图可以通过重复的将同一个函数交叉的应用在整个图像的子区域上，换句话说，通过使用一个线性分类器来对输入图像进行卷积，并增加一个偏置项，然后使用一个非线性函数来计算。假设我们在给定的层上第 k 个特征图为h^k，该特征图上的过滤器是由权重W^k 和偏置b_k 决定的 ，然后这个特征图h^k可以如下形式获得：

h^k_{ij} = \tanh ( (W^k * x)_{ij} + b_k ).

note：回顾下1D信号的卷积的定义 o[n] = f[n]*g[n] = \sum_{u=-\infty}^{\infty} f[u] g[n-u] = \sum_{u=-\infty}^{\infty} f[n-u] g[u]。这可以扩展到2D的形式： o[m,n] = f[m,n]*g[m,n] = \sum_{u=-\infty}^{\infty} \sum_{v=-\infty}^{\infty} f[u,v] g[m-u,n-v]。

    为了形成数据的更丰富的表征，每个隐藏层都是由多特征图组成的， \{h^{(k)}, k=0..K\}。一个隐藏层的权重W 可以被表示成4D张量的形式，其中包含了每个元素都是由目的特征图、源特征图、源垂直位置、源水平位置的组合而成的。偏置b 可以被表示成一个向量，其中包含着的每个元素都是对应着每个不同的目标特征图。图示的形式如下：

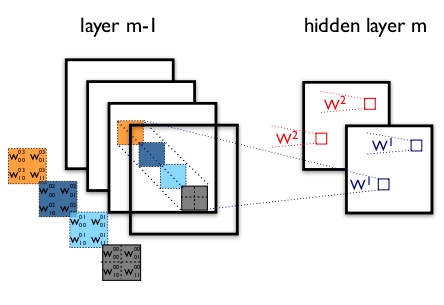


                                                                            图1：一个卷积层的例子

    该图显示的是一个CNN的两层。第m-1层包含着4个特征图。隐藏层m 包含着2个特征图（h^0 和 h^1）。在h^0 和 h^1（以蓝色和红色标出的方形区域）中的像素（神经元输出）是从第m-1层中的像素计算出来的，而第m-1层中的感受野为2×2（有色的矩形框）。注意这里感受野是如何跨越所有的四个输入特征图的。h^0 、h^1 的权重W^0 、 W^1所以是3D权重张量，其中第一个维度是用来索引输入特征图的，同时其他两个用来表示像素的坐标。

    将它们放在一起，那么，W^{kl}_{ij} 用来表示在第m 层的第k 个特征图上每个像素与第m-1层的第 I 个特征图的（i，j）位置上的像素相连的权重。

**五、卷积操作**

     在Theano中ConvOp是实现一个卷积层的主力。ConvOp通过theano.tensor.singal.conv2d来使用，这里需要两个符号输入：

* 一个 4D 张量对应着输入图像的一个mini-batch。 张量的原型为: [mini-batch size, 输入特征图的个数, 图像的高度, 图像的宽度].
* 一个 4D 张量对应着权重矩阵W. 该张量的原型为: [第m层特征图的个数， 第m-1层特征图的格式，过滤器的高度，过滤器的宽度]。

    下面是在theano中实现一个和图1一样的卷积层的代码。输入包含了3个特征图（一张RGB图），size为120×160.这里使用两个9×9的感受野的卷积过滤器：

1. **import** theano
2. **from** theano **import** tensor as T
3. **from** theano.tensor.nnet **import** conv
5. **import** numpy
7. rng = numpy.random.RandomState(23455)
9. # instantiate 4D tensor for input
10. input = T.tensor4(name='input')
12. # initialize shared variable for weights.
13. w\_shp = (2, 3, 9, 9)
14. w\_bound = numpy.sqrt(3 \* 9 \* 9)
15. W = theano.shared( numpy.asarray(
16. rng.uniform(
17. low=-1.0 / w\_bound,
18. high=1.0 / w\_bound,
19. size=w\_shp),
20. dtype=input.dtype), name ='W')
22. # initialize shared variable for bias (1D tensor) with random values
23. # IMPORTANT: biases are usually initialized to zero. However in this
24. # particular application, we simply apply the convolutional layer to
25. # an image without learning the parameters. We therefore initialize
26. # them to random values to "simulate" learning.
27. b\_shp = (2,)
28. b = theano.shared(numpy.asarray(
29. rng.uniform(low=-.5, high=.5, size=b\_shp),
30. dtype=input.dtype), name ='b')
32. # build symbolic expression that computes the convolution of input with filters in w
33. conv\_out = conv.conv2d(input, W)
35. # build symbolic expression to add bias and apply activation function, i.e. produce neural net layer output
36. # A few words on ``dimshuffle`` :
37. #   ``dimshuffle`` is a powerful tool in reshaping a tensor;
38. #   what it allows you to do is to shuffle dimension around
39. #   but also to insert new ones along which the tensor will be
40. #   broadcastable;
41. #   dimshuffle('x', 2, 'x', 0, 1)
42. #   This will work on 3d tensors with no broadcastable
43. #   dimensions. The first dimension will be broadcastable,
44. #   then we will have the third dimension of the input tensor as
45. #   the second of the resulting tensor, etc. If the tensor has
46. #   shape (20, 30, 40), the resulting tensor will have dimensions
47. #   (1, 40, 1, 20, 30). (AxBxC tensor is mapped to 1xCx1xAxB tensor)
48. #   More examples:
49. #    dimshuffle('x') -> make a 0d (scalar) into a 1d vector
50. #    dimshuffle(0, 1) -> identity
51. #    dimshuffle(1, 0) -> inverts the first and second dimensions
52. #    dimshuffle('x', 0) -> make a row out of a 1d vector (N to 1xN)
53. #    dimshuffle(0, 'x') -> make a column out of a 1d vector (N to Nx1)
54. #    dimshuffle(2, 0, 1) -> AxBxC to CxAxB
55. #    dimshuffle(0, 'x', 1) -> AxB to Ax1xB
56. #    dimshuffle(1, 'x', 0) -> AxB to Bx1xA
57. output = T.nnet.sigmoid(conv\_out + b.dimshuffle('x', 0, 'x', 'x'))
59. # create theano function to compute filtered images
60. f = theano.function([input], output)

用这个来做点有趣的事情：

1. **import** numpy
2. **import** pylab
3. **from** PIL **import** Image
5. # open random image of dimensions 639x516
6. img = Image.open(open('doc/images/3wolfmoon.jpg'))
7. # dimensions are (height, width, channel)
8. img = numpy.asarray(img, dtype='float64') / 256.
10. # put image in 4D tensor of shape (1, 3, height, width)
11. img\_ = img.transpose(2, 0, 1).reshape(1, 3, 639, 516)
12. filtered\_img = f(img\_)
14. # plot original image and first and second components of output
15. pylab.subplot(1, 3, 1); pylab.axis('off'); pylab.imshow(img)
16. pylab.gray();
17. # recall that the convOp output (filtered image) is actually a "minibatch",
18. # of size 1 here, so we take index 0 in the first dimension:
19. pylab.subplot(1, 3, 2); pylab.axis('off'); pylab.imshow(filtered\_img[0, 0, :, :])
20. pylab.subplot(1, 3, 3); pylab.axis('off'); pylab.imshow(filtered\_img[0, 1, :, :])
21. pylab.show()

应该会生成这样的输出：

注意到一个随机初始化的过滤器就像是一个边缘检测器！

    注意到我们使用了和MLP中一样的权重初始化公式。权重是从一个均匀分布 [-1/fan-in, 1/fan-in]中随机采样得到的。这里fan-in就是输入到一个隐藏单元的数量。对于MLPs来说，这是下层的单元的数量。对于CNN来说，需要考虑到输入特征图的数量和感受野的size。

**六、最大池化**

    另一个CNN的重要概念就是最大池化，这是非线性下采样的一种形式。最大池化是将输入图像划分成一个非重叠矩阵集合，然后对于每个子区域，输出他们的最大值。

    最大池化在视觉中很有用是基于以下两个原因：

1. 通过消除非最大值, 减少了上层的计算量.
2. 提供了一种平移不变性的形式. 想象下一个卷积层级联着一层最大池化层。对于一个单一的像素来说它有8个方向可以平移，如果是在一个2×2区域上使用最大池化，那么这8个可能的组合中的3个将会在卷积层上生成一样的输出，对于基于3×3的窗口上的最大池化来说，它达到了5/8（就是原来是3/8）。因为它提供额外的位置上的鲁棒性，最大池化是一种“明智”的方式来减少中间表征的维度。

    最大池化在theano中是通过theano.tensor.singal.downsample.max\_pool\_2d来实现的。这个函数输入为：N维张量的输入（N>=2），一个缩小因子。最大池化是在这个张量的最后2个维度上执行的操作。

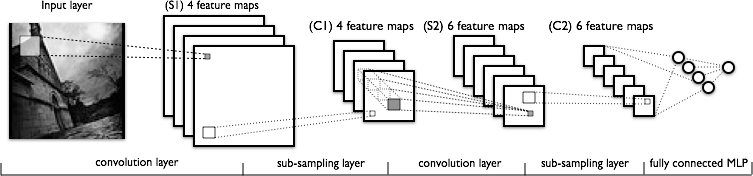
    一个例子胜过千言万语：

1. **from** theano.tensor.signal **import** downsample
3. input = T.dtensor4('input')
4. maxpool\_shape = (2, 2)
5. pool\_out = downsample.max\_pool\_2d(input, maxpool\_shape, ignore\_border=True)
6. f = theano.function([input],pool\_out)
8. invals = numpy.random.RandomState(1).rand(3, 2, 5, 5)
9. **print** 'With ignore\_border set to True:'
10. **print** 'invals[0, 0, :, :] =\n', invals[0, 0, :, :]
11. **print** 'output[0, 0, :, :] =\n', f(invals)[0, 0, :, :]
13. pool\_out = downsample.max\_pool\_2d(input, maxpool\_shape, ignore\_border=False)
14. f = theano.function([input],pool\_out)
15. **print** 'With ignore\_border set to False:'
16. **print** 'invals[1, 0, :, :] =\n ', invals[1, 0, :, :]
17. **print** 'output[1, 0, :, :] =\n ', f(invals)[1, 0, :, :]
18. This should generate the following output:
20. With ignore\_border set to True:
21. invals[0, 0, :, :] =
22. [[  4.17022005e-01   7.20324493e-01   1.14374817e-04   3.02332573e-01 1.46755891e-01]
23. [  9.23385948e-02   1.86260211e-01   3.45560727e-01   3.96767474e-01 5.38816734e-01]
24. [  4.19194514e-01   6.85219500e-01   2.04452250e-01   8.78117436e-01 2.73875932e-02]
25. [  6.70467510e-01   4.17304802e-01   5.58689828e-01   1.40386939e-01 1.98101489e-01]
26. [  8.00744569e-01   9.68261576e-01   3.13424178e-01   6.92322616e-01 8.76389152e-01]]
27. output[0, 0, :, :] =
28. [[ 0.72032449  0.39676747]
29. [ 0.6852195   0.87811744]]
31. With ignore\_border set to False:
32. invals[1, 0, :, :] =
33. [[ 0.01936696  0.67883553  0.21162812  0.26554666  0.49157316]
34. [ 0.05336255  0.57411761  0.14672857  0.58930554  0.69975836]
35. [ 0.10233443  0.41405599  0.69440016  0.41417927  0.04995346]
36. [ 0.53589641  0.66379465  0.51488911  0.94459476  0.58655504]
37. [ 0.90340192  0.1374747   0.13927635  0.80739129  0.39767684]]
38. output[1, 0, :, :] =
39. [[ 0.67883553  0.58930554  0.69975836]
40. [ 0.66379465  0.94459476  0.58655504]
41. [ 0.90340192  0.80739129  0.39767684]]

    注意到和大多数theano代码相比，max\_pool\_2d操作有一点特别。他需要一个缩小因子 ds （长度为2的元组，包含图像宽度和高度的缩小因子）在graph建立的时候需要知道的。这在将来也许会改变（也就是10年的这个theano和今年15年的这个函数有可能不一样，要注意）。

**七、完整的模型：LeNet**

    稀疏、卷积层和最大池化是LeNet模型家族的核心。不过这些模型的详细细节还是变化很大的，下图显示了一个LeNet模型的示意图：



    低层都是有交替的卷积和最大池化层构成的。高层是全连接层，对应着一个传统的MLP（隐藏层+逻辑回归）。输入到第一个全连接层的是低层的所有特征图的集合。

    从一个实现的角度来看，这意味着低层是在4D张量上操作的，然后平铺成一个2D矩阵栅格特征图，用来兼容之前的MLP实现。

**八、把上面的合并到一起**

    我们现在有了所有需要的。先来构建一个LeNetConvPoolLayer 类，用来实现{卷积+最大池化}层：

1. **class** LeNetConvPoolLayer(object):
2. """Pool Layer of a convolutional network """
4. **def** \_\_init\_\_(self, rng, input, filter\_shape, image\_shape, poolsize=(2, 2)):
5. """
6. Allocate a LeNetConvPoolLayer with shared variable internal parameters.
8. :type rng: numpy.random.RandomState
9. :param rng: a random number generator used to initialize weights
11. :type input: theano.tensor.dtensor4
12. :param input: symbolic image tensor, of shape image\_shape
14. :type filter\_shape: tuple or list of length 4
15. :param filter\_shape: (number of filters, num input feature maps,
16. filter height, filter width)
18. :type image\_shape: tuple or list of length 4
19. :param image\_shape: (batch size, num input feature maps,
20. image height, image width)
22. :type poolsize: tuple or list of length 2
23. :param poolsize: the downsampling (pooling) factor (#rows, #cols)
24. """
26. **assert** image\_shape[1] == filter\_shape[1]
27. self.input = input
29. # there are "num input feature maps \* filter height \* filter width"
30. # inputs to each hidden unit
31. fan\_in = numpy.prod(filter\_shape[1:])
32. # each unit in the lower layer receives a gradient from:
33. # "num output feature maps \* filter height \* filter width" /
34. #   pooling size
35. fan\_out = (filter\_shape[0] \* numpy.prod(filter\_shape[2:]) /
36. numpy.prod(poolsize))
37. # initialize weights with random weights
38. W\_bound = numpy.sqrt(6. / (fan\_in + fan\_out))
39. self.W = theano.shared(
40. numpy.asarray(
41. rng.uniform(low=-W\_bound, high=W\_bound, size=filter\_shape),
42. dtype=theano.config.floatX
43. ),
44. borrow=True
45. )
47. # the bias is a 1D tensor -- one bias per output feature map
48. b\_values = numpy.zeros((filter\_shape[0],), dtype=theano.config.floatX)
49. self.b = theano.shared(value=b\_values, borrow=True)
51. # convolve input feature maps with filters
52. conv\_out = conv.conv2d(
53. input=input,
54. filters=self.W,
55. filter\_shape=filter\_shape,
56. image\_shape=image\_shape
57. )
59. # downsample each feature map individually, using maxpooling
60. pooled\_out = downsample.max\_pool\_2d(
61. input=conv\_out,
62. ds=poolsize,
63. ignore\_border=True
64. )
66. # add the bias term. Since the bias is a vector (1D array), we first
67. # reshape it to a tensor of shape (1, n\_filters, 1, 1). Each bias will
68. # thus be broadcasted across mini-batches and feature map
69. # width & height
70. self.output = T.tanh(pooled\_out + self.b.dimshuffle('x', 0, 'x', 'x'))
72. # store parameters of this layer
73. self.params = [self.W, self.b]

     注意到当初始化权重值的时候，fan-in是由感受野的size和输入特征图的个数决定的。

    最后，使用在（[Theano3.3-练习之逻辑回归](http://blog.csdn.net/shouhuxianjian/article/details/46375461)）中的LogisticRegression 类和（[Theano3.4-练习之多层感知机](http://blog.csdn.net/shouhuxianjian/article/details/46400947)）中定义的HiddenLayer 类，我们可以如下来实例化：

1. x = T.matrix('x')   # the data is presented as rasterized images
2. y = T.ivector('y')  # the labels are presented as 1D vector of
3. # [int] labels
5. ######################
6. # BUILD ACTUAL MODEL #
7. ######################
8. **print** '... building the model'
10. # Reshape matrix of rasterized images of shape (batch\_size, 28 \* 28)
11. # to a 4D tensor, compatible with our LeNetConvPoolLayer
12. # (28, 28) is the size of MNIST images.
13. layer0\_input = x.reshape((batch\_size, 1, 28, 28))
15. # Construct the first convolutional pooling layer:
16. # filtering reduces the image size to (28-5+1 , 28-5+1) = (24, 24)
17. # maxpooling reduces this further to (24/2, 24/2) = (12, 12)
18. # 4D output tensor is thus of shape (batch\_size, nkerns[0], 12, 12)
19. layer0 = LeNetConvPoolLayer(
20. rng,
21. input=layer0\_input,
22. image\_shape=(batch\_size, 1, 28, 28),
23. filter\_shape=(nkerns[0], 1, 5, 5),
24. poolsize=(2, 2)
25. )
27. # Construct the second convolutional pooling layer
28. # filtering reduces the image size to (12-5+1, 12-5+1) = (8, 8)
29. # maxpooling reduces this further to (8/2, 8/2) = (4, 4)
30. # 4D output tensor is thus of shape (batch\_size, nkerns[1], 4, 4)
31. layer1 = LeNetConvPoolLayer(
32. rng,
33. input=layer0.output,
34. image\_shape=(batch\_size, nkerns[0], 12, 12),
35. filter\_shape=(nkerns[1], nkerns[0], 5, 5),
36. poolsize=(2, 2)
37. )
39. # the HiddenLayer being fully-connected, it operates on 2D matrices of
40. # shape (batch\_size, num\_pixels) (i.e matrix of rasterized images).
41. # This will generate a matrix of shape (batch\_size, nkerns[1] \* 4 \* 4),
42. # or (500, 50 \* 4 \* 4) = (500, 800) with the default values.
43. layer2\_input = layer1.output.flatten(2)
45. # construct a fully-connected sigmoidal layer
46. layer2 = HiddenLayer(
47. rng,
48. input=layer2\_input,
49. n\_in=nkerns[1] \* 4 \* 4,
50. n\_out=500,
51. activation=T.tanh
52. )
54. # classify the values of the fully-connected sigmoidal layer
55. layer3 = LogisticRegression(input=layer2.output, n\_in=500, n\_out=10)
57. # the cost we minimize during training is the NLL of the model
58. cost = layer3.negative\_log\_likelihood(y)
60. # create a function to compute the mistakes that are made by the model
61. test\_model = theano.function(
62. [index],
63. layer3.errors(y),
64. givens={
65. x: test\_set\_x[index \* batch\_size: (index + 1) \* batch\_size],
66. y: test\_set\_y[index \* batch\_size: (index + 1) \* batch\_size]
67. }
68. )
70. validate\_model = theano.function(
71. [index],
72. layer3.errors(y),
73. givens={
74. x: valid\_set\_x[index \* batch\_size: (index + 1) \* batch\_size],
75. y: valid\_set\_y[index \* batch\_size: (index + 1) \* batch\_size]
76. }
77. )
79. # create a list of all model parameters to be fit by gradient descent
80. params = layer3.params + layer2.params + layer1.params + layer0.params
82. # create a list of gradients for all model parameters
83. grads = T.grad(cost, params)
85. # train\_model is a function that updates the model parameters by
86. # SGD Since this model has many parameters, it would be tedious to
87. # manually create an update rule for each model parameter. We thus
88. # create the updates list by automatically looping over all
89. # (params[i], grads[i]) pairs.
90. updates = [
91. (param\_i, param\_i - learning\_rate \* grad\_i)
92. **for** param\_i, grad\_i **in** zip(params, grads)
93. ]
95. train\_model = theano.function(
96. [index],
97. cost,
98. updates=updates,
99. givens={
100. x: train\_set\_x[index \* batch\_size: (index + 1) \* batch\_size],
101. y: train\_set\_y[index \* batch\_size: (index + 1) \* batch\_size]
102. }
103. )

    这里没有实际训练和早期停止的代码，因为它实际上和MLP的一样。感兴趣的读者可以访问DeepLearningTutorials中“code”这个文件夹。

**九、运行该代码**

    用户可以如下形式运行该代码：

python code/convolutional\_mlp.py

    接下来的输出可以在 Core i7-2600K CPU clocked at 3.40GHz上使用默认参数和flags ‘floatX=float32’:来得到：

Optimization complete.

Best validation score of 0.910000 % obtained at iteration 17800,with test

performance 0.920000 %

The code **for** file convolutional\_mlp.py ran **for** 380.28m

    使用GeForce GTX 285，得到如下结果：

Optimization complete.

Best validation score of 0.910000 % obtained at iteration 15500,with test

performance 0.930000 %

The code **for** file convolutional\_mlp.py ran **for** 46.76m

    使用GeForce GTX 480的结果：

Optimization complete.

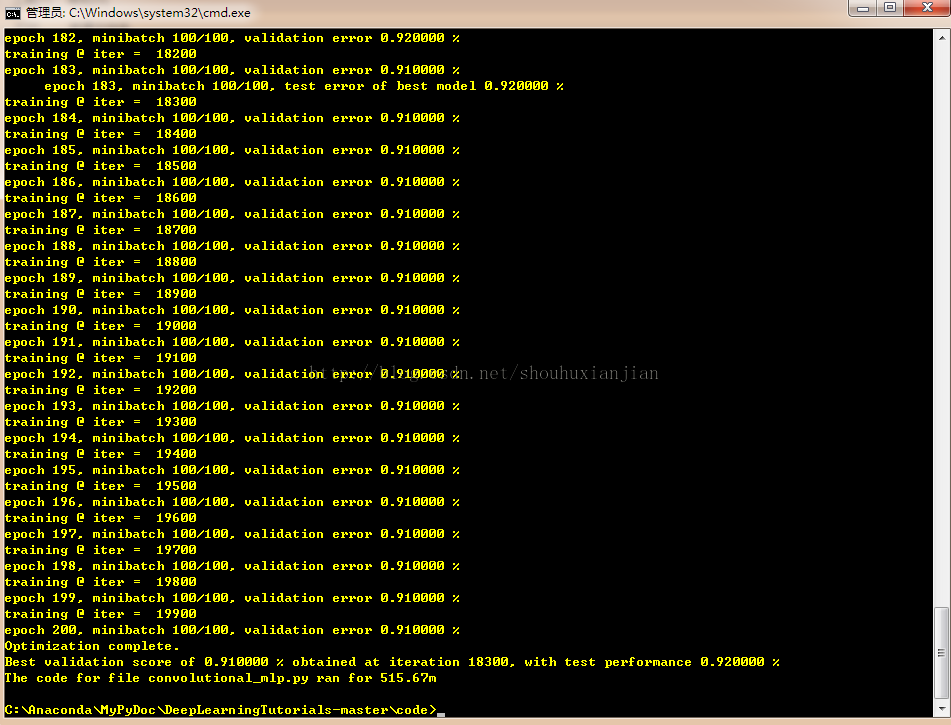
Best validation score of 0.910000 % obtained at iteration 16400,with test

performance 0.930000 %

The code **for** file convolutional\_mlp.py ran **for** 32.52m

    注意到在验证的时候和测试时候误差的差异（迭代的次数），这是因为硬件中舍入机制的不同实现造成的。这可以被忽略掉，不用管。

在win7\_64bit+cuda6.5\_64bit+anaconda2.1.0\_64bit+gtx 780ti,结果：



（不知道为什么时间反而多了，原因待分析）。

**十、提示和技巧**

**选择超参数**

    CNN训练的时候特别需要技巧，因为它们相比一个标准的MLP来说有着更多的超参数。不过通常的学习率和正则化约束的经验规则还是适用的，接下来就是在优化CNN的时候需要记住的。

**过滤器的数量**

    当选择每一层的过滤器的个数的时候，记得一个单一的卷积过滤器的激活值的计算比传统的MLPs代价更高昂。

    假设层(l-1)包含着K^{l-1}个特征图和M \times N个像素位置（即，位置个数乘以特征图个数），在形状为m \times n的层l 上有K^l 个过滤器。然后计算一个特征图（在所有过滤器能够使用的(M-m) \times (N-n) 个像素位置上使用一个 m \times n的过滤器）的代价为 (M-m) \times (N-n) \times m \times n \times K^{l-1}。总的cost是K^l 乘以这个值。如果在同一层的所有的特征不是连接到之前一层的所有特征，那么事情就会变得更加的复杂。

    对于一个标准的MLP，cost将会只是 K^l \times K^{l-1}，这里 在l层上有K^l个不同的神经元。同样的，在CNNs中使用的过滤器的个数通常要小于在MLPs中隐藏单元的个数，并且依赖于特征图的size（它本身的一个输入图像的size和过滤器形状的函数）

    因为特征图的size随着深度的增加而下降，靠近输入层的层会有着更少的过滤器而更高层就会有更多的过滤器。事实上，为了在每一层中平衡下计算量，这些特征个数和像素位置个数的乘积通常在层之间是差不多保持稳定的。为了保留有关输入的信息，将会需要维持激活的总数（特征图的数量乘以像素位置数量）来使得从这一层到下一层的时候没有减少（当然，我们希望在做有监督训练的时候没有变得更少）。特征图的个数直接控制着能力（capacity），同样依赖于可利用的样本的个数和任务的复杂程度。

**过滤器的形状**

    一般过滤器形状在文献中变化万千，通常是基于特定的数据集的。在MNIST-sized 图像（28×28）这样的上，最好的结果通常是第一层上有着5×5的过滤器大小，同时对于自然图像数据集（通常在每一维上有着上百个像素）倾向于第一层使用更大的过滤器，例如12×12或者15×15。

    所以这里的技巧就是在给定的数据集的基础上，去找到正确的“粒度”（即，过滤器形状），从而能够在合适的尺寸下生成好的抽象表征。

**最大池化的形状**

    通常来说值为2×2或者没有最大池化操作。非常大的输入图像也许在低层上会有着4×4的池化。不过记得，这会以因子为16来减少信号的维度，同时也许会导致丢失过多的信息。

脚注

[1] 更清晰的说，我们使用“unit”或者“neuron”来表示人工神经元，“cell”来表示生物神经元。

**提示**

    如果你想在一个新的数据集上使用这个模型，这里有一些提示也许有助于你生成更好的结果：

* 对数据进行白化 (e.g. with PCA)
* 在每个epoch上衰减学习率

**参考资料：**

[1] 官网：http://deeplearning.net/tutorial/lenet.html#lenet

[2] Deep learning with Theano 官方中文教程（翻译）（四） 卷积神经网络（CNN）：http://www.cnblogs.com/charleshuang/p/3651843.html