[theano中的一些备忘](http://blog.csdn.net/silence1214/article/details/44096237)

2015-03-06 10:38 132人阅读 [评论](http://blog.csdn.net/silence1214/article/details/44096237#comments)(0) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/silence1214/article/details/44096237#report)

上个学期一直想用theano，但是苦于当时文档读的不是很懂，并且很多东西感觉不到咋弄的，同时时间比较紧急，就没怎么弄。这个学习来，发现文档大部分都看懂了，并且的确这玩意很好，单不说可以利用GPU，就是自动求导我就很喜欢的很。

昨天在写function的时候，结合共享变量来写遇到点问题，今天早上结合错误信息，搞了一下，发现还是有一些规律的，先举个例子吧

如果存在以下代码：

**[python]** [view plaincopyprint?](http://blog.csdn.net/silence1214/article/details/44096237)[在CODE上查看代码片](https://code.csdn.net/snippets/613038)

1. **import** numpy,theano,theano.tensor as T

**[python]** [view plaincopyprint?](http://blog.csdn.net/silence1214/article/details/44096237)[在CODE上查看代码片](https://code.csdn.net/snippets/613038)

1. x = T.dmatrix('x')

**[python]** [view plaincopyprint?](http://blog.csdn.net/silence1214/article/details/44096237)[在CODE上查看代码片](https://code.csdn.net/snippets/613038)

1. w = T.dmatrix('w')

**[python]** [view plaincopyprint?](http://blog.csdn.net/silence1214/article/details/44096237)[在CODE上查看代码片](https://code.csdn.net/snippets/613038)

1. init\_x = numpy.asarray([1], theano.config.floatX)

**[python]** [view plaincopyprint?](http://blog.csdn.net/silence1214/article/details/44096237)[在CODE上查看代码片](https://code.csdn.net/snippets/613038)

1. init\_w = numpy.asarray([2], theano.config.floatX)

**[python]** [view plaincopyprint?](http://blog.csdn.net/silence1214/article/details/44096237)[在CODE上查看代码片](https://code.csdn.net/snippets/613038)

1. y = x\*w

**[python]** [view plaincopyprint?](http://blog.csdn.net/silence1214/article/details/44096237)[在CODE上查看代码片](https://code.csdn.net/snippets/613038)



**[python]** [view plaincopyprint?](http://blog.csdn.net/silence1214/article/details/44096237)[在CODE上查看代码片](https://code.csdn.net/snippets/613038)

1. f = theano.function([w,x], y)

对于上述代码，需要注意的地方是使用符号变量对x和w的定义中的类型以及对x和w的值的初始化的类型。

定义的时候是一个matrix，此时对应的初始化的值就应该是个matrix的类型，而在初始化中，传入的是一个vector，如果使用

**[python]** [view plaincopyprint?](http://blog.csdn.net/silence1214/article/details/44096237)[在CODE上查看代码片](https://code.csdn.net/snippets/613038)

1. init\_x = numpy.asarray(1)

来定义，那么这个数就是scalar了，所以会出错的，报错的信息就是类似于shape()出来的值和定义的不一样。所以这个地方务必注意下，如果定义符号变量是矩阵就按照矩阵来初始化他们的值，这个地方应该用这样的公式来初始化：

**[python]** [view plaincopyprint?](http://blog.csdn.net/silence1214/article/details/44096237)[在CODE上查看代码片](https://code.csdn.net/snippets/613038)

1. init\_x = numpy.asarray([[1]], theano.config.floatX)
2. init\_y = numpy.asarray([[2]], theano.config.floatX)

注意下中括号的个数，就理解意思了。

接下来再说的是，如果w是共享变量怎么办。

在theano.function的输入参数中，共享变量是不能作为它的输入的。但是可以参与运算，是什么意思呢？就是先看下面的代码吧

**[python]** [view plaincopyprint?](http://blog.csdn.net/silence1214/article/details/44096237)[在CODE上查看代码片](https://code.csdn.net/snippets/613038)

1. x = T.dmatrix('x)
2. init\_w = numpy.asarray([[2]], theano.config.floatX)
3. w = theano.shared(init\_w, theano.config.floatX, 'w')

**[python]** [view plaincopyprint?](http://blog.csdn.net/silence1214/article/details/44096237)[在CODE上查看代码片](https://code.csdn.net/snippets/613038)

1. init\_x = numpy.asarray([[1]], theano.config.floatX)

**[python]** [view plaincopyprint?](http://blog.csdn.net/silence1214/article/details/44096237)[在CODE上查看代码片](https://code.csdn.net/snippets/613038)

1. y = x\*w

**[python]** [view plaincopyprint?](http://blog.csdn.net/silence1214/article/details/44096237)[在CODE上查看代码片](https://code.csdn.net/snippets/613038)

1. f = theano.function([x], y)

此时输入参数只有x，此时就是对的了，因为x不是共享变量，但是y的计算是有共享变量x的参与的。

再说一个此时的 updates参数干啥用

接着上面的代码补充一下下面几句代码

**[python]** [view plaincopyprint?](http://blog.csdn.net/silence1214/article/details/44096237)[在CODE上查看代码片](https://code.csdn.net/snippets/613038)

1. y\_val = y.mean() # 此时也可以用T.mean(y)
2. gw = T.cost(y\_val, [w])

**[python]** [view plaincopyprint?](http://blog.csdn.net/silence1214/article/details/44096237)[在CODE上查看代码片](https://code.csdn.net/snippets/613038)

1. f = theano.function([x], y, updates=[(w, w-gw)])

可以看出来多个个updates参数，意思就是在计算出来y的时候，同时也进行一下updates里面的操作，让w=w-gw。而gw此时尽管不被显式的计算，但是既然有这个操作在，系统也会去计算gw，进而操作下updates里面的含义。

再补充一个地方就是求导的时候，记得一定是标量对其它的求导，也就是y求出来之后来个平均值或者直接求和，这样就是个标量了

数据类型的地方一定注意好，否则一大堆错误，看懂错误信息还好，看不懂就无从下手了。

# [theano的一个备注1](http://blog.csdn.net/silence1214/article/details/40406859)

2014-10-23 22:47 234人阅读 [评论](http://blog.csdn.net/silence1214/article/details/40406859#comments)(0) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/silence1214/article/details/40406859#report)

theano中的function，如果有updates参数的话  
updates的参数必须是成对出现的，这一对中包含共享变量和表达式，也可以是一个dictionary，key是共享变量，value是表达式。它的意思是不管什么时候这个函数运行，表达式的值将取代共享变量的值。  
  
  
也就是说function可以任意定义关系，但是如果涉及到updates参数的话，必须是和共享变量有关的

# [theano 2](http://blog.csdn.net/silence1214/article/details/40581503)

2014-10-29 09:47 400人阅读 [评论](http://blog.csdn.net/silence1214/article/details/40581503#comments)(0) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/silence1214/article/details/40581503#report)

1：已经可以自己写完MLP的例子了，昨天跑MLP的时候遇到个问题，为了追踪问题，出错的时候给出了2个配置选项

exception\_verbosity=high  
optimizer=fast\_compile

添加上这2个配置选项之后，的确可以具体的定位到出错的位置了，但是很奇怪不知道怎么回事，和网上说的结果不一样，我把网上实例代码下载下来也是不行，晚上找了个哥们让他帮跑了下，很奇怪，一样的代码和数据集合他就跑出来了，后来经过对比发现是这2个配置选项带来的错误，尤其是后者带来的，我给开发小组发了个邮件，theano的作者回复了我一下：

Some optimizations that improve numerical stability are not included in  
the "fast\_compile" optimizer, so it may be that the gradient updates are  
not precise enough to get out of a plateau or local minimum. It is also  
possible that the optimization diverges, and that NaN values appear in  
the parameters.

经过验证，的确是出现了NaN的情况。所以这个问题也就解决了

2：由于theano中不带小数点的都当做整数处理了，所以在计算类似5/6的时候要小心，给出的结果肯定是0，因为都是整数，结果也还是整数了。如果想精确计算就这样5./6也就是先把5转换成为float的类型了，于是结果也就带小数点了，这个地方也坑了我很久，因为我要初始化一个参数，精度在5/a a要大于5的 结果这样初始化出来就是0了。所以必须写成5./a

3：整体感觉theano方便，因为不需要自己去求导了，只要记得起之前的cost就好了，但是可能之前写matlab写习惯了，总觉得这种方式不知道靠谱不。其实还是靠谱的

4：theano中由于对公式是进行编译的，不像matlab还可以在函数过程中打印当前的输入和输出，在theano中可没这么方便，但是theano提供了方法的。需要theano专门的方法。

# [theano 3](http://blog.csdn.net/silence1214/article/details/40651729)

2014-10-31 11:24 239人阅读 [评论](http://blog.csdn.net/silence1214/article/details/40651729#comments)(0) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/silence1214/article/details/40651729#report)

1：不得不说theano的理解，真的和一般的不一样，可能是符号化编程就是这样子的吧，要习惯要适应

2：目前最有疑问的问题就是类似如下

cost = a.cost(x,y)

index = T.lscaler()

train\_model = function([index], cost, updates=updates, givens={x:train\_set[index:index+batch\_size]})

也就是为啥function中要那么取代x，而function中出现了index，没出现x

其实是这样子的，之前定义了cost 就是纯粹的函数和自变量的关系。而在function中是把这个函数编译为theano的自己的表示方式，那么怎么表示呢。就是要取代自变量x的，如果x是符号变量的话，而取代x的话 你可以直接写train\_set来替代function中的index，但是我不想这么做，我要输入个小的batch，那么怎么做？那么就在这里我再引入一个针对于取train \_set中某一部分数据的变量，那就是index，其实index是为x服务的，x是为cost服务的。

# [thean 4](http://blog.csdn.net/silence1214/article/details/40658437)

2014-10-31 17:59 161人阅读 [评论](http://blog.csdn.net/silence1214/article/details/40658437#comments)(0) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/silence1214/article/details/40658437#report)

今天基本耗费了大半天来写代码用theano，但是到最后写完编译的时候，遇到一大堆的错误，才知道。原来把theano用错了，写的时候就觉得这个思路不太行的通。我先举个例子吧。我一个模型，里面有训练、测试和验证。

我自己先定义了个NN，竟然把三个集合都传递进去了，使用了真正的数据集合去测试的，经过看到的编译错误才周到，这都错了

应该定义模型的时候，只需要在模型类的内部传入需要参与运算的变量即可，这个是符号变量，在外面训练的时候回被代替的，我举个例子

class NN:

    def \_\_init\_\_(x, y):

       self.x = x

       self.y =y

   def cost(self):

       xxxxx

   def errors(self):

      xxxxx

这个地方，x，y你只需要当做是符号就行了。外面怎么传递给他，外面说了算的。我最开始直接把x，y的实际数据集合传递进来了，才发现不对。

之前觉得theano很难理解，现在看来也不是那么回事，还是有规律的

[theano中训练方法和模型的一些写法](http://blog.csdn.net/silence1214/article/details/44155089)

2015-03-09 15:18 261人阅读 [评论](http://blog.csdn.net/silence1214/article/details/44155089#comments)(0) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/silence1214/article/details/44155089#report)

按照这theano的tutorial开始跟着写了，因为去年年底之前学习过一段时间，但是当时时间少，并且很多地方也没搞懂，大多都是看着书来模仿，结果出错不好找地方之外，自己如果根据自己的想法随便写下结果就出错了。这几天好好的学习了下。我来总结下

在softmax或者逻辑回归的代码中：

1：在教程中写法是，先写一个类，在类的init方法中初始化w和b，以及计算概率。

2：然后分别再另外的函数中计算cost和error。

3：在类外来进行训练。

以上方法由于在init方法中，要进行概率计算，所以在初始化类的时候至少要传递进去x，在计算cost和error的时候要传递进去y。所以如果此时训练方法写在类内的话就不行。因为数据也是在训练方法内生成的，在类别没办法传递进去（此时，你也许会说在类外面单独加载数据怎么样？因为训练方法内其实传递进去的是数据的索引，所以这个在外面单独加载数据的话也不太现实。所以加载数据还是和训练方法写在一起是最方便的）

为了解决以上问题，很容易，就是把init方法中，把在类外面不能传递的参数，全部去掉，然后在别的计算的地方用到的话，再传递进去。比如把概率预测拿出来，写成一个函数，把x传递进去。然后其余的保持不变。这样在训练方法内，因为有对x和y的声明，以及对应数据的传递进去，所以不会出问题

我这里写好了一个，发上来作为备忘吧，为下一步的更高层次的封装做准备：

**[python]** [view plaincopyprint?](http://blog.csdn.net/silence1214/article/details/44155089)[在CODE上查看代码片](https://code.csdn.net/snippets/615016)

1. **import** numpy, theano, theano.tensor as T, gzip, cPickle
3. **class** NN():
5. **def** \_\_init\_\_(self, n\_in, n\_out):
6. self.w = theano.shared(numpy.asarray(numpy.zeros([n\_in, n\_out]), theano.config.floatX))
7. self.b = theano.shared(numpy.asarray(numpy.zeros(n\_out), theano.config.floatX))
8. **def** get\_probalblity(self, x):
9. **return**  T.nnet.softmax(T.dot(x, self.w) + self.b)
10. **def** get\_prediction(self, x, y):
11. **return** T.argmax(self.get\_probalblity(x), 1)
12. **def** cost(self, x, y):
13. p\_y\_given\_x = self.get\_probalblity(x)
14. **return**  -T.mean(T.log(p\_y\_given\_x[T.arange(y.shape[0]), y]))
15. **def** error(self, x, y):
16. prediction = self.get\_prediction(x, y)
17. **return** T.mean(T.neq(prediction, y))
18. **def** load\_data(self):
19. f = gzip.open('mnist.pkl.gz')
20. trainxy, validatexy, testxy = cPickle.load(f)
21. **def** share\_data(xy):
22. x,y = xy
23. x = theano.shared(numpy.asarray(x, theano.config.floatX))
24. y = theano.shared(numpy.asarray(y, theano.config.floatX))
25. **return** [x, T.cast(y, 'int32')]
26. trainx, trainy = share\_data(trainxy)
27. validatex,validatey = share\_data(validatexy)
28. testx, testy = share\_data(testxy)
29. **return** [(trainx,trainy),(validatex,validatey),(testx,testy)]
30. **def** train(self):
31. x = T.matrix('x', theano.config.floatX)
32. y = T.ivector('y')
33. [(trainx,trainy),(validatex,validatey),(testx,testy)] = self.load\_data()
34. gw,gb = T.grad(self.cost(x,y), [self.w, self.b])
35. index = T.lscalar()
37. batch\_size = 600
38. trainModel = theano.function([index], self.cost(x,y), updates=[(self.w, self.w-0.13\*gw), (self.b, self.b-0.13\*gb)], givens={x:trainx[index\*batch\_size:(index+1)\*batch\_size], y:trainy[index\*batch\_size:(index+1)\*batch\_size]})
39. validateModel = theano.function([index], self.error(x,y), givens={x:validatex[index\*batch\_size:(index+1)\*batch\_size], y:validatey[index\*batch\_size:(index+1)\*batch\_size]})
40. testModel = theano.function([index], self.error(x,y), givens={x:testx[index\*batch\_size:(index+1)\*batch\_size], y:testy[index\*batch\_size:(index+1)\*batch\_size]})
42. best\_validate\_error = numpy.Inf
43. best\_test\_error = 0
44. patience = 5000
45. increasement = 2
46. train\_batchs = trainx.get\_value().shape[0]/batch\_size
47. validate\_batchs = validatex.get\_value().shape[0]/batch\_size
48. test\_batchs = testx.get\_value().shape[0]/batch\_size
49. validate\_frequency = min(patience/2, train\_batchs)
50. epochs = 1000
51. epoch = 1
52. ite = 0
53. stopping = False
54. **while** (epoch < epochs) **and** (**not** stopping):
55. **for** i **in** xrange(train\_batchs):
56. ite += 1
57. this\_cost = trainModel(i)
58. **if** ite%validate\_frequency == 0:
59. this\_validate\_error = numpy.mean([validateModel(j) **for** j **in** xrange(validate\_batchs)])
60. **print** ('ite:%d/%d, cost:%f, validate:%f'%(ite, epoch, this\_cost, this\_validate\_error))
61. **if** this\_validate\_error < best\_validate\_error:
62. **if** this\_validate\_error < 0.995\*best\_validate\_error:
63. patience = max(patience, ite\*increasement)
64. this\_test\_error = numpy.mean([testModel(j) **for** j **in** xrange(test\_batchs)])
65. best\_validate\_error = this\_validate\_error
66. best\_test\_error = this\_test\_error
67. **print** ('ite:%d/%d, cost:%f, validate:%f, test:%f'%(ite, epoch, this\_cost, this\_validate\_error, this\_test\_error))
68. **if** patience <= ite:
69. stopping = True
70. **break**
71. epoch +=1
72. **print** ('best validate error:%f, best test error:%f'%(best\_validate\_error, best\_test\_error))

75. **if** \_\_name\_\_ == '\_\_main\_\_':
76. nn = NN(784, 10)
77. nn.train()

以上训练结果，和tutorial给出的基本一致。

。。。。。

ite:5810/70, cost:0.329380, validate:0.075104  
ite:5810/70, cost:0.329380, validate:0.075104, test:0.075000  
ite:5893/71, cost:0.329054, validate:0.075208  
ite:5976/72, cost:0.328735, validate:0.075104  
ite:5976/72, cost:0.328735, validate:0.075104, test:0.075104  
ite:6059/73, cost:0.328422, validate:0.075000  
ite:6059/73, cost:0.328422, validate:0.075000, test:0.074896  
ite:6142/74, cost:0.328116, validate:0.074792  
ite:6142/74, cost:0.328116, validate:0.074792, test:0.074896  
best validate error:0.074792, best test error:0.074896

[theano做的MLP](http://blog.csdn.net/silence1214/article/details/44176995)

2015-03-10 16:27 150人阅读 [评论](http://blog.csdn.net/silence1214/article/details/44176995#comments)(0) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/silence1214/article/details/44176995#report)

在贡献一个，和tutorial写的不太一样，现在是在学习着写。这个结果可以达到1.68%的误差，需要200多万次的迭代，快1000个epoch了

**[python]** [view plaincopyprint?](http://blog.csdn.net/silence1214/article/details/44176995)[在CODE上查看代码片](https://code.csdn.net/snippets/615919)

1. **import** theano, numpy, theano.tensor as T, gzip, cPickle, time

4. **class** HiddenLayer():
5. **def** \_\_init\_\_(self, n\_in, n\_out, x, w=None, b=None):
6. **if** w **is** None:
7. t = numpy.sqrt(6.0/(n\_in+n\_out))
8. w = theano.shared(numpy.asarray(numpy.random.uniform(-t, t, [n\_in, n\_out]), theano.config.floatX))
9. self.w = w
10. **if** b **is** None:
11. b = theano.shared(numpy.asarray(numpy.zeros(n\_out), theano.config.floatX))
12. self.b = b
13. self.out = T.tanh(T.dot(x, self.w) + self.b)
14. self.params = [self.w, self.b]
15. **def** get\_l1(self):
16. **return** T.sum(T.abs\_(self.w))
17. **def** get\_l2(self):
18. **return** T.sum(self.w\*\*2)
20. **class** MLP():
21. **def** \_\_init\_\_(self, n\_in=784, n\_out=10, n\_hidden=500):
22. self.n\_in = n\_in
23. self.n\_out = n\_out
24. self.n\_hidden = n\_hidden
25. self.w = theano.shared(numpy.asarray(numpy.zeros([n\_hidden, n\_out]), theano.config.floatX))
26. self.b = theano.shared(numpy.asarray(numpy.zeros(n\_out), theano.config.floatX))
27. self.params = [self.w, self.b]
28. **def** p\_y\_given\_x(self, x):
29. **return** T.nnet.softmax(T.dot(x, self.w) + self.b)
30. **def** pred(self, x):
31. **return** T.argmax(self.p\_y\_given\_x(x), 1)
32. **def** cost(self, x, y):
33. p\_y\_given\_x = self.p\_y\_given\_x(x)
34. **return** -T.mean(T.log(p\_y\_given\_x[T.arange(y.shape[0]), y]))
35. **def** error(self, x, y):
36. pred = self.pred(x)
37. **return** T.mean(T.neq(pred, y))
38. **def** get\_l1(self):
39. **return** T.sum(T.abs\_(self.w))
40. **def** get\_l2(self):
41. **return** T.sum(self.w\*\*2)
43. **def** train(self):
44. **def** load\_data():
45. f = gzip.open('mnist.pkl.gz')
46. trainxy, validatexy, testxy = cPickle.load(f)
47. **def** share\_data(xy):
48. x,y = xy
49. x = theano.shared(numpy.asarray(x, theano.config.floatX))
50. y = theano.shared(numpy.asarray(y, theano.config.floatX))
51. **return** [x, T.cast(y, 'int32')]
52. trainx,trainy = share\_data(trainxy)
53. validatex,validatey = share\_data(validatexy)
54. testx,testy = share\_data(testxy)
55. **return** [(trainx, trainy), (validatex, validatey), (testx, testy)]
56. [(trainx, trainy), (validatex, validatey), (testx, testy)] = load\_data()
57. **print** 'Load the data successfully...'
58. batch\_size = 20
59. train\_batch = trainx.get\_value().shape[0]/batch\_size
60. validate\_batch = validatex.get\_value().shape[0]/batch\_size
61. test\_batch = testx.get\_value().shape[0]/batch\_size
63. x = T.matrix('x', theano.config.floatX)
64. y = T.ivector('y')
65. n\_in = self.n\_in
66. n\_hidden = self.n\_hidden
67. n\_out = self.n\_out
68. l1\_decay = 0.0
69. l2\_decay = 0.0001
70. hl = HiddenLayer(n\_in, n\_hidden, x)
72. mlp = MLP(n\_hidden, n\_out)
73. cost = mlp.cost(hl.out, y) + l1\_decay\*(mlp.get\_l1() + hl.get\_l1()) + l2\_decay\*(mlp.get\_l2() + hl.get\_l2())
74. error = mlp.error(hl.out, y)
75. params = hl.params + mlp.params
76. grad\_params = [T.grad(cost, param) **for** param **in** params]
77. updates = [(param, param-0.01\*grad) **for** param,grad **in** zip(params, grad\_params)]
79. index = T.lscalar()
80. trainModel = theano.function([index], cost, updates=updates, givens={x:trainx[index\*batch\_size:(index+1)\*batch\_size], y:trainy[index\*batch\_size:(index+1)\*batch\_size]})
81. validateModel = theano.function([index], error, givens={x:validatex[index\*batch\_size:(index+1)\*batch\_size], y:validatey[index\*batch\_size:(index+1)\*batch\_size]})
82. testModel = theano.function([index], error, givens={x:testx[index\*batch\_size:(index+1)\*batch\_size], y:testy[index\*batch\_size:(index+1)\*batch\_size]})
84. patience = 5000
85. frequency = min(patience/2, train\_batch)
86. increase = 2
87. best\_validate\_error = numpy.Inf
88. best\_test\_error = 0
89. epochs = 1000
90. epoch = 1
91. ite = 0
92. stopping = False
94. **while** (epoch < epochs) **and** (**not** stopping):
95. **for** i **in** xrange(train\_batch):
96. ite += 1
97. this\_cost = trainModel(i)
98. **if** ite%frequency == 0:
99. this\_validate\_error = numpy.mean([validateModel(j) **for** j **in** xrange(validate\_batch)])
100. **print** 'ite:%d/%d, cost:%f, validate error:%f'%(ite, epoch, this\_cost, this\_validate\_error)
101. **if** this\_validate\_error < best\_validate\_error:
102. **if** this\_validate\_error < 0.995\*best\_validate\_error:
103. patience = max(patience, ite\*increase)
104. best\_validate\_error = this\_validate\_error
105. best\_test\_error = numpy.mean([testModel(j) **for** j **in** xrange(test\_batch)])
106. **print** 'ite:%d/%d, cost:%f, validate error:%f, test error:%f'%(ite, epoch, this\_cost, this\_validate\_error, best\_test\_error)
107. **if** patience <= ite:
108. stopping = True
109. **break**
110. epoch += 1
112. **print** 'best validate error:%f, best test error:%f'%(best\_validate\_error, best\_test\_error)
114. **if** \_\_name\_\_=='\_\_main\_\_':
115. mlp = MLP()
116. start\_time = time.clock()
117. mlp.train()
118. end\_time = time.clock()
119. **print** 'total consuming %d mins'%((end\_time-start\_time)/60.0)

# [对theano求导的时候的一个疑问](http://blog.csdn.net/silence1214/article/details/44236501)

2015-03-13 09:10 123人阅读 [评论](http://blog.csdn.net/silence1214/article/details/44236501#comments)(0) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/silence1214/article/details/44236501#report)

假定我们有下面下面4个参数w1,w2,b1,b2以及对应的cost。我看到早些版本的求导是这么写的

params1 = [w1,b1]

params2 = [w2,b2]

params = params1.extend(params2)

gparams = [(T.grad(cost, param) for param in params)]

还有一种是

gparams = T.grad(cost, params)

我个人觉得是没有区别

[theano中的numpy.random和theano.tensor中的RandomStream](http://blog.csdn.net/silence1214/article/details/44402895)

2015-03-18 09:52 121人阅读 [评论](http://blog.csdn.net/silence1214/article/details/44402895#comments)(0) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/silence1214/article/details/44402895#report)

1：最开始这2个一直没搞懂具体的使用的地方，这两天写了一些代码，突然想到一个实验可以证明一下他们的区别，于是一实验，还真对比出来了他们的异同点了呢。先看实验吧

2：实验代码：

第一个实验：

**[python]** [view plaincopyprint?](http://blog.csdn.net/silence1214/article/details/44402895)[在CODE上查看代码片](https://code.csdn.net/snippets/622137)

1. x1 = T.lscalar()
2. y1 = T.lscalar()
3. y1 = x1 + numpy.random.randint(10)
4. f1 = theano.function([x1], y1)
5. **for** i **in** xrange(10):
6. **print** f1(1)

第二个实验：

**[python]** [view plaincopyprint?](http://blog.csdn.net/silence1214/article/details/44402895)[在CODE上查看代码片](https://code.csdn.net/snippets/622137)

1. rng = RandomStreams(1024)
2. x2 = T.lscalar()
3. y2 = T.ivector()
4. y2 = rng.random\_integers([1,1], 0, 10)
5. f2 = theano.function([], y2)
6. **for** i **in** xrange(10):
7. **print** f2()

实验结果：第一个实验的结果是一致输出一样的数字，而第二个输出的是都不一样的结果。这样区分的很明显了。

因为theano中对function都是编译起来的，真正在运算的时候，才把对应的 自变量给替换掉去运算。所以对于这个的解释就是

因为第一个实验在编译的时候，就把numpy.random,randint的值给算出来了，编译进去了。所以不管运行多少次，结果肯定一样了。

第二个实验，我猜测是编译进去了一个随机函数，然后每次运行的时候，就调用的还是随机函数，于是结果不太一样。