使用GPU和Theano加速深度学习

发表于2015-08-09 22:09| 2018次阅读| 来源DominoDatalab| 3 条评论| 作者Manojit Nandi

[深度学习](http://www.csdn.net/tag/%E6%B7%B1%E5%BA%A6%E5%AD%A6%E4%B9%A0/news)[机器学习](http://www.csdn.net/tag/%E6%9C%BA%E5%99%A8%E5%AD%A6%E4%B9%A0/news)[Python](http://www.csdn.net/tag/python/news)

**摘要：**Theano是主流的深度学习Python库之一，亦支持GPU，然而Theano入门较难，Domino的这篇博文介绍了如何使用GPU和Theano加速深度学习，教程从多层感知器到卷积神经网络，由浅入深，是不错的入门资料。

**【编者按】**GPU因其浮点计算和矩阵运算能力有助于加速深度学习是业界的共识，Theano是主流的深度学习Python库之一，亦支持GPU，然而Theano入门较难，Domino的这篇博文介绍了如何使用GPU和Theano加速深度学习，使用更简单的基于Theano的 [Nolearn](https://github.com/dnouri/nolearn)库。教程由多层感知器及卷积神经网络，由浅入深，是不错的入门资料。

**基于Python的深度学习**

实现神经网络算法的Python库中，最受欢迎的当属Theano。然而，Theano并不是严格意义上的神经网络库，而是一个Python库，它可以实现各种各样的数学抽象。正因为如此，Theano有着陡峭的学习曲线，所以我将介绍基于Theano构建的有更平缓的学习曲线的两个神经网络库。

第一个库是 [Lasagne](https://github.com/Lasagne/Lasagne)。该库提供了一个很好的抽象，它允许你构建神经网络的每一层，然后堆叠在彼此的顶部来构建一个完整的模型。尽管这比Theano显得更好，但是构建每一层，然后附加在彼此顶部会显得有些冗长乏味，所以我们将使用 [Nolearn](https://github.com/dnouri/nolearn)库，它在Lasagne库上提供了一个类似 [Scikit-Learn](http://scikit-learn.org/stable/)风格的API，能够轻松地构建多层神经网络。

**延伸阅读**： [从Theano到Lasagne：基于Python的深度学习的框架和库](http://www.csdn.net/article/2015-08-01/2825362)

由于这些库默认使用的不是Domino硬件，所以你需要创建一个requirements.txt文件，该文件内容如下：

-e git://github.com/Theano/Theano.git#egg=Theano

-e git://github.com/lasagne/lasagne.git#egg=lasagne

nolearn==0.5.0

**配置Theano**

现在，在我们导入Lasagne库和Nolearn库之前，首先我们需要配置Theano，使其可以使用GPU硬件。要做到这一点，我们需要在我们的工程目录中新建一个.theanorc文件，该文件内容如下：

[global] device = gpu floatX = float32 [nvcc] fastmath = True

这个.theanorc文件必须放置在主目录中。在你的本地计算机上，这个操作可以手工完成，但我们不能直接访问Domino机器的主目录，所以我们需要使用下面的代码将文件移到它的主目录中：

import os import shutil destfile = "/home/ubuntu/.theanorc" open(destfile,

'a').close() shutil.copyfile(".theanorc", destfile)

上面的代码会在主目录创建了一个空的.theanorc文件，然后复制我们项目目录下的.theanorc文件内容到该文件中。

将硬件切换到GPU后，我们可以来做一下测试，使用Theano文档中提供的测试代码来看看Theano是否能够检测到GPU。

from theano import function, config, shared, sandbox

import theano.tensor as T

import numpy

import time

vlen = 10 \* 30 \* 768 # 10 x #cores x # threads per core

iters = 1000

rng = numpy.random.RandomState(22)

x = shared(numpy.asarray(rng.rand(vlen), config.floatX))

f = function([], T.exp(x))

print f.maker.fgraph.toposort()

t0 = time.time()

for i in xrange(iters):

r = f()

t1 = time.time()

print 'Looping %d times took' % iters, t1 - t0, 'seconds'

print 'Result is', r

if numpy.any([isinstance(x.op, T.Elemwise) for x in f.maker.fgraph.toposort()]):

print 'Used the cpu'

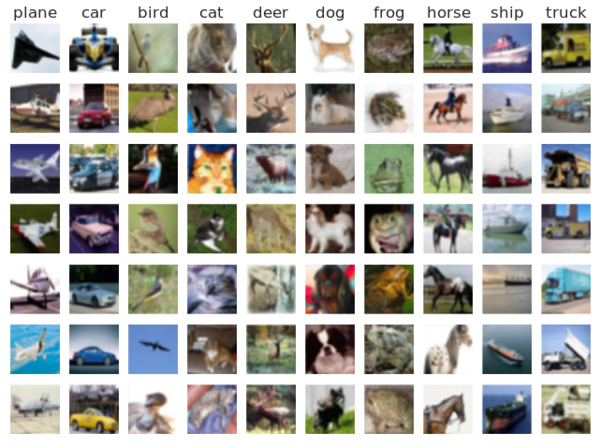
else:

print 'Used the gpu'

如果Theano检测到GPU，上面的函数运行时间应该需要0.7秒，并且输出“Used the gpu”。否则，整个过程将需要2.6秒的运行时间，同时输出“Used the cpu”'。如果输出的是后一个，那么你肯定是忘记将硬件切换到GPU了。

**数据集**

对于这个项目，我们将使用CIFAR-10图像数据集，它来自10个不同的类别，包含了60000个32x32大小的彩色图像。



幸运的是，这些数据属于 [pickled](https://en.wikipedia.org/wiki/Pickle_(Python))格式，所以我们可以使用辅助函数来加载数据，将每个文件加载到NumPy数组中并返回训练集（Xtr），训练集标签（Ytr），测试集（Xte）以及测试集标签（Yte）。下列代码归功于 [Stanford's CS231n](http://cs231n.stanford.edu/)课程的工作人员。

import cPickle as pickle

import numpy as np

import os

def load\_CIFAR\_file(filename):

'''Load a single file of CIFAR'''

with open(filename, 'rb') as f:

datadict= pickle.load(f)

X = datadict['data']

Y = datadict['labels']

X = X.reshape(10000, 3, 32, 32).transpose(0,2,3,1).astype('float32')

Y = np.array(Y).astype('int32')

return X, Y

def load\_CIFAR10(directory):

'''Load all of CIFAR'''

xs = []

ys = []

for k in range(1,6):

f = os.path.join(directory, "data\_batch\_%d" % k)

X, Y = load\_CIFAR\_file(f)

xs.append(X)

ys.append(Y)

Xtr = np.concatenate(xs)

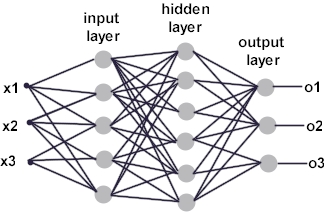
Ytr = np.concatenate(ys)

Xte, Yte = load\_CIFAR\_file(os.path.join(directory, 'test\_batch'))

return Xtr, Ytr, Xte, Yte

**多层感知器**

多层感知器是一种最简单的神经网络模型。该模型包括一个输入层数据，一个施加一些数学变换的隐藏层，以及一个输出层用来产生一个标签（不管是分类还是回归，都一样）。



图片来源：http://dms.irb.hr/tutorial/tut\_nnets\_short.php

在我们使用训练数据之前，我们需要把它的灰度化，把它变成一个二维矩阵。此外，我们将每个值除以255然后减去0.5。当我们对图像进行灰度化时，我们将每一个（R,G,B）元组转换成0到255之间的浮点值）。通过除以255，可以标准化灰度值映射到[0,1]之间。接下来，我们将所有的值减去0.5，映射到区间[ -0.5，0.5 ]上。现在，每个图像都由一个1024维的数组表示，每一个值都在- 0.5到0.5之间。在训练分类网络时，标准化你的输入值在[-1,1]之间是个很常见的做法。

X\_train\_flat = np.dot(X\_train[...,:3], [0.299, 0.587, 0.114]).reshape(X\_train.shape[0],-1).astype(np.float32)

X\_train\_flat = (X\_train\_flat/255.0)-0.5

X\_test\_flat = np.dot(X\_test[...,:3], [0.299, 0.587, 0.114]).reshape(X\_test.shape[0],-1).astype(np.float32)

X\_test\_flat = (X\_test\_flat/255.0)-.5

使用nolearn的API，我们可以很容易地创建一个输入层，隐藏层和输出层的多层感知器。hidden\_num\_units = 100表示我们的隐藏层有100个神经元，output\_num\_units = 10则表示我们的输出层有10个神经元，并与标签一一对应。输出前，网络使用 [softmax](https://en.wikipedia.org/wiki/Softmax_function)函数来确定最可能的标签。迭代50次并且设置verbose=1来训练模型，最后会输出每次迭代的结果及其需要的运行时间。

net1 = NeuralNet( layers = [ ('input', layers.InputLayer), ('hidden',

layers.DenseLayer), ('output', layers.DenseLayer), ], #layers parameters:

input\_shape = (None, 1024), hidden\_num\_units = 100, output\_nonlinearity

= softmax, output\_num\_units = 10, #optimization parameters: update = nesterov\_momentum,

update\_learning\_rate = 0.01, update\_momentum = 0.9, regression = False,

max\_epochs = 50, verbose = 1, )

从侧面来说，这个接口使得它很容易建立深层网络。如果我们想要添加第二个隐藏层，我们所需要做的就是把它添加到图层参数中，然后在新增的一层中指定多少个神经元。

net1 = NeuralNet( layers = [ ('input', layers.InputLayer), ('hidden1',

layers.DenseLayer), ('hidden2', layers.DenseLayer), #Added Layer Here ('output',

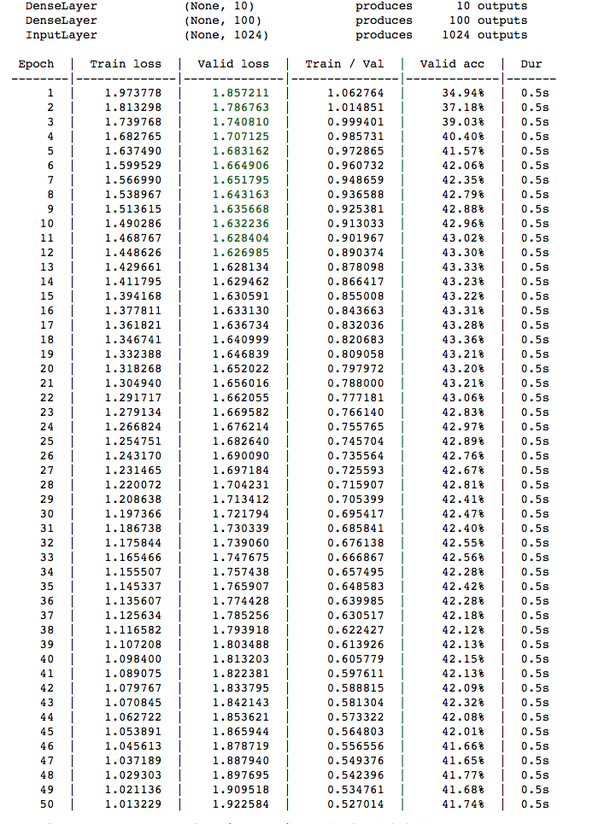
layers.DenseLayer), ], #layers parameters: input\_shape = (None, 1024),

hidden1\_num\_units = 100, hidden2\_num\_units = 100, #Added Layer Params Here

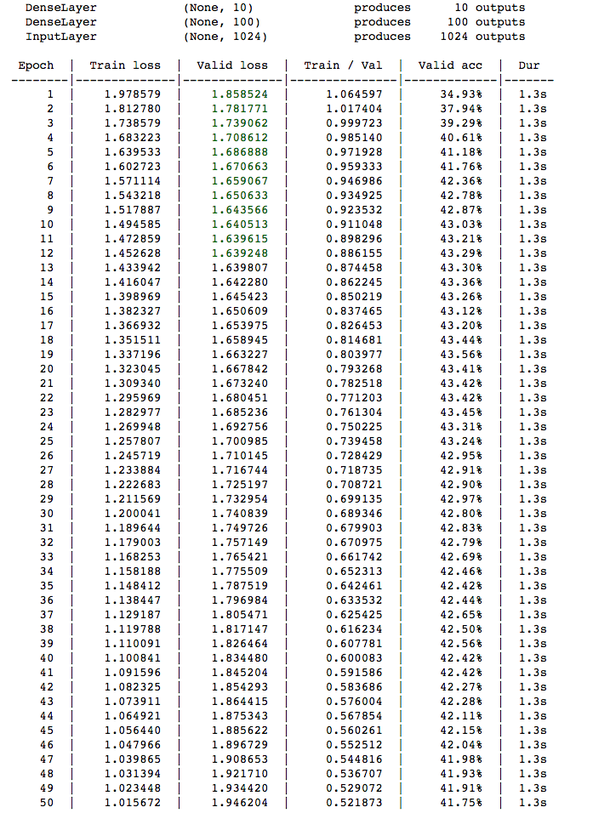
现在，正如我前面提到的关于Nolearn类似Scikit-Learn风格的API，我们可以用fit函数来拟合神经网络。

net1.fit(X\_train\_flat, y\_train)

当网络使用GPU训练时，我们可以看到每次迭代时间通常需要0.5秒。



另一方面，当Domino的硬件参数设置为XX-Large（32 core, 60 GB RAM），每次迭代时间通常需要1.3秒。



通过GPU训练的神经网络，我们可以看到在训练网络上大约提速了3倍。正如预期的那样，使用GPU训练好的神经网络和使用CPU训练好的神经网络产生了类似的结果。两者产生了相似的测试精度（约为41%）以及相似的训练损失。

通过下面代码，我们可以在测试数据上测试网络：

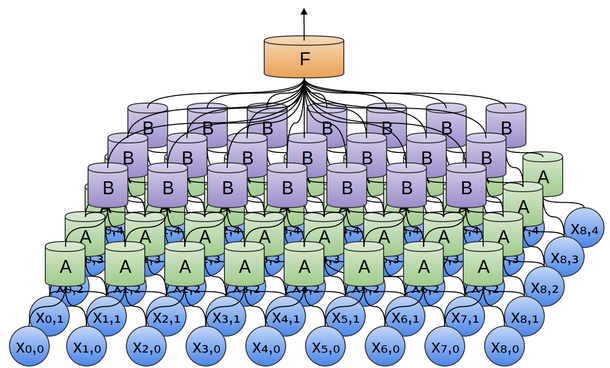
y\_pred1 = net1.predict(X\_test\_flat)

print "The accuracy of this network is: %0.2f" % (y\_pred1 == y\_test).mean()

最后，我们在测试数据上得到的精度为41%。

**卷积网络**

卷积神经网络是一种更为复杂的神经网络结构，它的一个层中的神经元和上一层的一个子集神经元相连。结果，卷积往往会池化每个子集的输出。



图片来源： <http://colah.github.io/posts/2014-07-Conv-Nets-Modular/>

卷积神经网络在企业和 [Kaggle 竞赛](http://jeffreydf.github.io/diabetic-retinopathy-detection/)中很受欢迎，因为它能灵活地学习不同的问题并且易扩展。

同样，在我们建立卷积神经网络之前，我们首先必须对数据进行灰度化和变换。这次我们会保持图像32x32的大小不变。此外，我已经修改了矩阵的行顺序，所以每个图像现在被表示为（color，x，y）格式。跟之前一样，我将特征的每个值除以255，再减去0.5，最后将数值映射到区间（-1，1）。

X\_train\_2d = np.dot(X\_train[...,:3], [0.299, 0.587, 0.114]).reshape(-1,1,32,32).astype(np.float32)

X\_train\_2d = (X\_train\_2d/255.0)-0.5

X\_test\_2d = np.dot(X\_test[...,:3], [0.299, 0.587, 0.114]).reshape(-1,1,32,32).astype(np.float32)

X\_train\_2d = (X\_train\_2d/255.0)-0.5

现在我们可以构造卷积神经网络了。该网络由输入层，3个卷积层，3个2x2池化层，200个神经元隐藏层以及最后的输出层构成。

net2 = NeuralNet(

layers = [

('input', layers.InputLayer),

('conv1', layers.Conv2DLayer),

('pool1', layers.MaxPool2DLayer),

('conv2', layers.Conv2DLayer),

('pool2', layers.MaxPool2DLayer),

('conv3', layers.Conv2DLayer),

('pool3', layers.MaxPool2DLayer),

("hidden4", layers.DenseLayer),

("output", layers.DenseLayer),

],

#layer parameters:

input\_shape = (None, 1, 32, 32),

conv1\_num\_filters = 16, conv1\_filter\_size = (3, 3), pool1\_pool\_size = (2,2),

conv2\_num\_filters = 32, conv2\_filter\_size = (2, 2) , pool2\_pool\_size = (2,2),

conv3\_num\_filters = 64, conv3\_filter\_size = (2, 2), pool3\_pool\_size = (2,2),

hidden4\_num\_units = 200,

output\_nonlinearity = softmax,

output\_num\_units = 10,

#optimization parameters:

update = nesterov\_momentum,

update\_learning\_rate = 0.015,

update\_momentum = 0.9,

regression = False,

max\_epochs = 5,

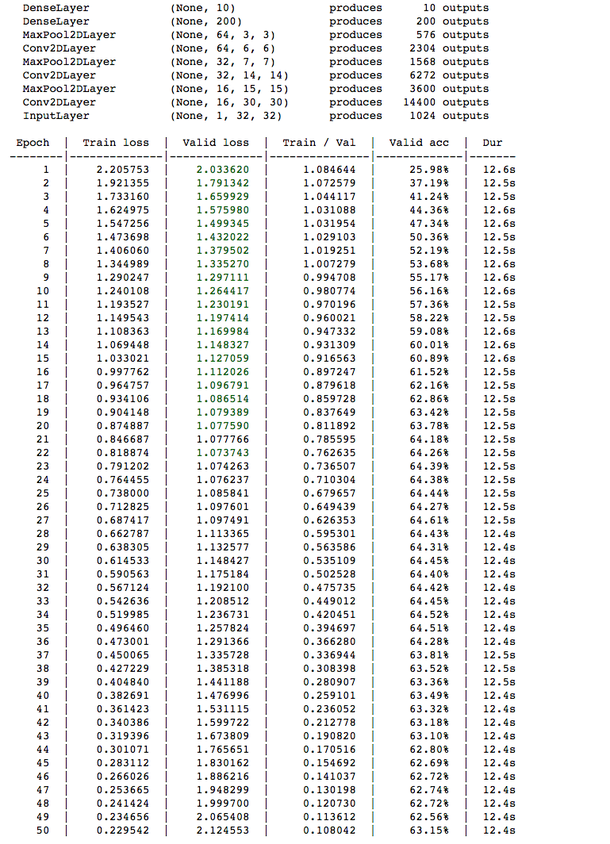
verbose = 1,

)

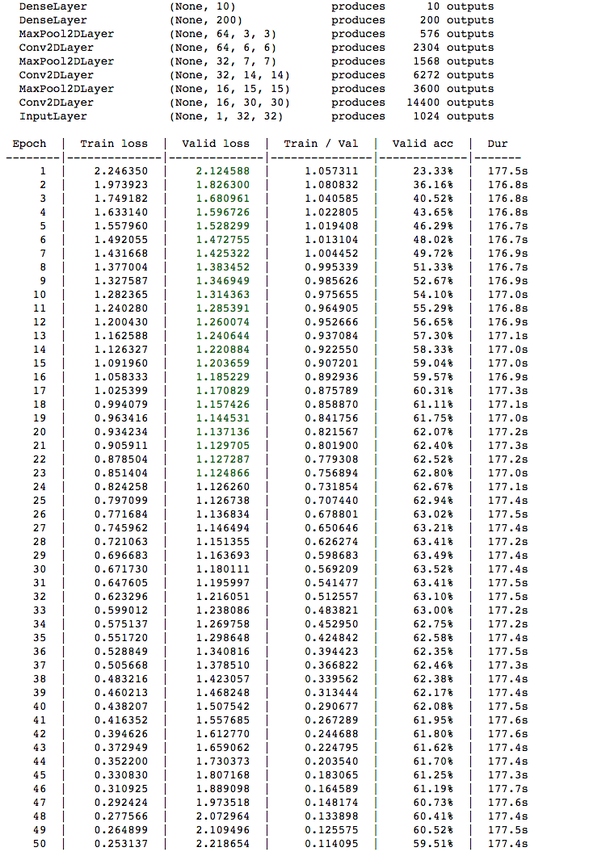
接着，我们再次使用fit函数来拟合模型。

net2.fit(X\_train\_2d, y\_train)

与多层感知器相比，卷积神经网络的训练时间会更长。使用GPU来训练，大多数的迭代需要12.8s来完成，然而，卷积神经网络验证损失约为63%，超过了验证损失为40%的多层感知器。也就是说，通过卷积层和池化层的结合，我们可以提高20%的精度。



在只有Domino的XX-大型硬件层的CPU上，每个训练周期大概需要177秒完成，接近于3分钟。也就是说，用GPU训练，训练时间提升了大约15倍。



和前面一样，我们可以看到在CUP上训练的卷积神经网络与GPU上训练的卷积神经网络有着类似的结果，相似的验证精度与训练损失。

此外，当我们在测试数据上测试卷积神经网络时，我们得到了61%的精度。

y\_pred2 = net2.predict(X\_test\_2d)

print "The accuracy of this network is: %0.2f" % (y\_pred2 == y\_test).mean()

建立卷积神经网络的所有代码都可以在ConvolutionNN.py这个 [文件](https://app.dominodatalab.com/LeJit/GPU_Example/view/ConvolutionNN.py)中找到。

最后，正如你所看到的，使用GPU训练的深度神经网络会加快运行加速，在这个项目中它提升的速度在3倍到15倍之间。无论是在工业界还是学术界，我们经常会使用多个GPU，因为这会大大减少深层网络训练的运行时间，通常能从几周下降至几天。

原文链接： [Faster deep learning with GPUs and Theano](http://blog.dominodatalab.com/gpu-computing-and-deep-learning/)（译者/刘帝伟 审校/刘翔宇、朱正贵 责编/周建丁）

**关于译者：**  [刘帝伟](http://blog.csdn.net/dream_angel_z)，中南大学软件学院在读研究生，关注机器学习、数据挖掘及生物信息领域。