LINEAR EQUATION REPORT

School or Department: Wu Han University

Grade and Specialty: Excellent engineer 2016

Name: Guo Yang, Kong Chuishun, Tang Rui

Advisor: Fangling Pu

May 7 2018

【Abstract】 In this lab, we consider vector-valued linear and affine functions, and systems of linear equations. And we use a python program to familiar it. In the application, we introduce a method to train Binarized Neural Networks (BNNs) - neural networks with binary weights and activations.

【Key words】 linear equation; BNN, Multidimensional linear function model

Table of Contents

Chapter 1 Introduction

1.1 Linear and affine functions

1.1.1Vector-valued functions of vectors.

The notation f : $R^n \rightarrow R^m$ means that *f* is a function that maps real n-vectors to real m-vectors. The value of the function *f*, evaluated at an n-vector x, is an m-vector f(x) = ($f_{1(x)}, f_{2(x)}, \ldots, f_{m(x)}$). Each of the components *fi* of *f* is itself a scalar-valued function of *x*. As with scalar-valued functions, we sometimes write *f(x) = f(x1,x2,...,xn)* to emphasize that f is a function of n scalar arguments. We use the same notation for each of the components of f, writing $fi(x) = fi(x_1, x_2, \ldots, x_n)$ to emphasize that fi is a function mapping the scalar arguments $x_1, \ldots, x_n$ into a scalar.

1.1.2 The matrix-vector product function

Suppose A is an m×n matrix. We can define a function f : $R^n \rightarrow R^m$ by *f(x) = Ax*. The inner product function *f : Rn* $\rightarrow$ *R*, defined as $f(x) = a^T x$, , is the special case with m = 1.

1.1.3 Superposition and linearity.

The function *f : Rn* $\rightarrow$ *Rm*, defined by *f(x) = Ax*, is linear, i.e., it satisfies the superposition property:

$$f(\alpha x + \beta y) = \alpha f(x) + \beta f(y) \tag{1.1}$$

holds for all n-vectors x and y and all scalars α and β. It is a good exercise to parse this simple looking equation, since it involves overloading of notation. On the left-hand side, the scalar-vector multiplications αx and βy involve n-vectors, and the sum αx+βy is the sum of two n-vectors. The function f maps n-vectors to m-vectors, so $f(\alpha x+\beta y)$ is an m-vector. On the right-hand side, the scalar-vector multiplications and the sum are those for m-vectors. Finally, the equality sign is equality between two m-vectors.

We can verify that superposition holds for f using properties of matrix-vector and scalar-vector multiplication:

$$\begin{aligned} f(\alpha x + \beta y) &= A(\alpha x + \beta y) \\ &= A(\alpha x) + A(\beta y) \\ &= \alpha(Ax) + \beta(Ay) \\ &= \alpha f(x) + \beta f(y) \end{aligned}$$

Thus we can associate with every matrix *A* a linear function *f(x) = Ax.*

The converse is also true. Suppose f is a function that maps n-vectors to m vectors, and is linear, i.e., (1.1) holds for all n-vectors x and y and all scalars α and β. Then there exists an m×n matrix A such that *f(x) = Ax* for all x. This can be shown in the same way as for scalar-valued functions, by showing that if f is linear, then

$$f(x) = x_1 f_{(e1)} + x_2 f_{(e2)} + \cdots + x_n f_{(en)} \qquad 1.2$$

where e_k is the kth unit vector of size n. The right-hand side can also be written as a matrix-vector product Ax, with

$$A = [\, f_{(e1)}\, f_{(e2)} \cdots f_{(en)}]$$

It is easily shown that the matrix-vector representation of a linear function is unique. If f : $R^n \rightarrow R^m$ is a linear function, then there exists exactly one matrix A such that f(x) = Ax for all x.

1.1.4 Examples of linear functions

In the examples below we define functions *f* that map n-vectors *x* to *n*-vectors *f(x)*. Each function is described in words, in terms of its effect on an arbitrary x. In each case we give the associated matrix multiplication representation.

• Negation. f changes the sign of *x: f(x) = −x.*

Negation can be expressed as *f(x) = Ax* with *A = −I.*

• Reversal. f reverses the order of the elements of *x: f(x) = (xn,xn−1,...,x1).* The reversal function can be expressed as *f(x) = Ax* with

$$A = \begin{pmatrix} 0 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \cdots & 0 \end{pmatrix}$$

(This is the n×n identity matrix with the order of its columns reversed.

1.1.5 Affine functions.

A vector-valued function f : $R_n \rightarrow R_m$ is called affine if it can be expressed as f(x) = Ax + b, where A is an m×n matrix and b is an m-vector. It can be shown that a function f : Rn → Rm is affine if and only if

$$f(\alpha x + \beta y) = \alpha f(x) + \beta f(y)$$

holds for all n-vectors x, y, and all scalars α, β that satisfy $\alpha + \beta = 1$. In other words, superposition holds for affine combinations of vectors. (For linear functions, superposition holds for any linear combinations of vectors.) The matrix A and the vector b in the representation of an affine function as f(x) = Ax + b are unique. These parameters can be obtained by evaluating f at the vectors 0, *e1,...,en*, where *ek* is the kth unit vector in Rn. We have

$$A = [\ f(e1)-f(0)\ f(e2)-f(0) \cdots f(en)-f(0)\ ], \quad b = f(0).$$

Just like affine scalar-valued functions, affine vector-valued functions are often called linear, even though they are linear only when the vector b is zero.

1.2 Systems of linear equations

Consider a set (also called a system) of m linear equations in n variables or unknowns$x_1, \dots \dots x_n$,

$$\begin{aligned} A_{11}x_1 + A_{12}x_2 + \cdots + A_{1n}x_n &= b_1 \\ A_{21}x_1 + A_{22}x_2 + \cdots + A_{2n}x_n &= b_2 \\ &\ \ \vdots \\ A_{m1}x_1 + A_{m2}x_2 + \cdots + A_{mn}x_n &= b_m \end{aligned}$$

The numbers A_{ij} are called the coefficients in the linear equations, and the numbers b_i are called the right-hand sides (since by tradition, they appear on the right-hand side of the equation). These equations can be written succinctly in matrix notation as

$$Ax = b$$

In this context, the $m \times n$ matrix A is called the *coefficient matrix,* and the m-vector b is called the *right-hand* side. An n-vector x is called a *solution* of the linear equations if $Ax = b$ holds. A set of linear equations can have no solutions, one solution, or multiple solutions.

The set of linear equations is called over-determined if $m > n$, under-determined if $m < n$, and square if $m = n$; these correspond to the coefficient matrix being tall, wide, and square, respectively. When the system of linear equations is over-determined, there are more equations than variables or unknowns. When the system of linear equations is under-determined, there are more unknowns than equations. When the system of linear equations is square, the numbers of unknowns and equations is the same. A set of equations with zero right-hand side, $Ax = 0$, is called a homogeneous set of equations. Any homogeneous set of equations has $x = 0$ as a solution.

Base on the basic conception, we have several usages. For example, *Leontief input-output model.*

We consider an economy with n different industrial sectors. We let x_i be the economic activity level, or total production output, of sector i, for $i = 1, \ldots, n$, measured in a common unit, such as (billions of) dollars. The output of each sector flows to other sectors, to support their production, and also to consumers. We denote the total consumer demand for sector i as d_i, for $i = 1, \ldots, n$. Supporting the output level xj for sector j requires $A_{ij}x_j$ output for sector i.

We refer to $A_{ij}x_j$ as the sector i input that flows to sector i. (We can have $A_{ii} \neq 0$; for example, it requires some energy to support the production of energy.) Thus, $A_{i1}x_1 + A_{i2}x_2 + \cdots + A_{in}x_n$ is the total sector i output required by, or flowing into, the n industrial sectors. The matrix A is called the input-output matrix of the economy, since it describes the flows of sector outputs to the inputs of itself and other sectors. The vector A_x gives the sector outputs required to support the production levels given by x.

Chapter 2　Our experiment

2.1 Brief Introduction

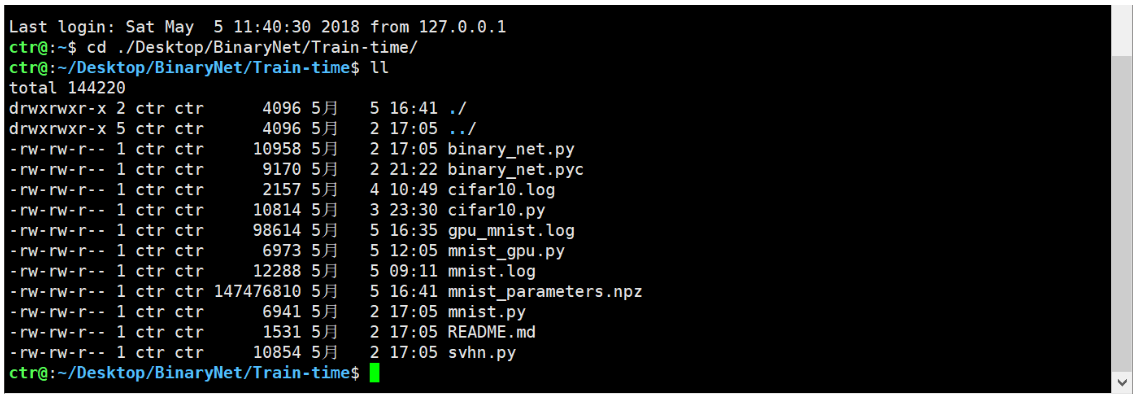
Based on what we have learned and what we three had done in DSP Laboratory, we want to establish a multidimensional linear function model to train *Cifar10* dataset using binarized neural network.

Cifar10 is a collection of data sets for universal object recognition. It is an advanced science project research institute led by the Canadian government. *Cifar10* consists of 60, 000 32*32 RGB color images in 10 categories. 50,000 training, 10,000 tests (cross validation). The biggest feature of this dataset is that it migrates to universal objects and applies to multiple categories.

2.2 Specific operation

As follow figure, we login DSP laboratory by Xshell5 and have installed our lab environment including annocoda2.7, pylearn2, lasagna, Theano, Cuda and Pycuda. It cost me two days to config all environment well and that tortures me a lot.

Anyway, I successfully solve those problems. Finally we can run our machine learning code using the tesla k80 GPU. Following are our notes about this lab code(train cifar10 dataset using binarized neural network).



```
Last login: Sat May  5 11:40:30 2018 from 127.0.0.1
ctr@:~$ cd ./Desktop/BinaryNet/Train-time/
ctr@:~/Desktop/BinaryNet/Train-time$ ll
total 144220
drwxrwxr-x 2 ctr ctr      4096 5月   5 16:41 ./
drwxrwxr-x 5 ctr ctr      4096 5月   2 17:05 ../
-rw-rw-r-- 1 ctr ctr     10958 5月   2 17:05 binary_net.py
-rw-rw-r-- 1 ctr ctr      9170 5月   2 21:22 binary_net.pyc
-rw-rw-r-- 1 ctr ctr      2157 5月   4 10:49 cifar10.log
-rw-rw-r-- 1 ctr ctr     10814 5月   3 23:30 cifar10.py
-rw-rw-r-- 1 ctr ctr     98614 5月   5 16:35 gpu_mnist.log
-rw-rw-r-- 1 ctr ctr      6973 5月   5 12:05 mnist_gpu.py
-rw-rw-r-- 1 ctr ctr     12288 5月   5 09:11 mnist.log
-rw-rw-r-- 1 ctr ctr 147476810 5月   5 16:41 mnist_parameters.npz
-rw-rw-r-- 1 ctr ctr      6941 5月   2 17:05 mnist.py
-rw-rw-r-- 1 ctr ctr      1531 5月   2 17:05 README.md
-rw-rw-r-- 1 ctr ctr     10854 5月   2 17:05 svhn.py
ctr@:~/Desktop/BinaryNet/Train-time$ 
```

```
######################################################
#
# 结构:
#
# 输入层
#    |
# 卷积层 1 + BN 层 + 激活层
#    |
# 卷积层 2 + 最大值混合层 2 + BN 层 + 激活层   (binary)
#    |
# 卷积层 3 + BN 层 + 激活层
#    |
# 卷积层 4 + 最大值混合层 4 + BN 层 + 激活层
#    |
# 卷积层 5 + BN 层 + 激活层
#    |
# 卷积层 6 + 最大值混合层 6 + BN 层 + 激活层
#    |
# 全连接层 1 + BN 层 + 激活层  (binary)
#    |
# 全连接层 2 + BN 层 + 激活层
#    |
# 全连接层 3 + BN 层
#    |
# 输出
#
######################################################
```

```

    # theano.function(inputs, outputs=None, mode=None, updates=None, givens=None):
    #   根据输入 inputs 计算数据 outputs 的函数，其中:
    #   inputs:  列表类型，用来保存输入量
    #   outputs: 列表或字典类型，用来保存输出量。输入量与输出量的映射关系通常在输出量的定义中体现。
    #   updates: 一组可迭代更新的量(shared_variable, new_expression),
    #       对其中的 shared_variable 输入用 new_expression 表达式更新
    #
    # lasagne.updates.adam(loss_or_grads, params, learning_rate): 用于参数更新,
    #   其中: loss_or_grads: 误差或梯度
    #   params:             要更新的参数
    #   learning_rate:      更新速率(学习速率)
    # lasagne.layers.get_output(layer):             对指定网络，计算网络输出;
    # lasagne.objectives.categorical_crossentropy(predictions, targets): 计算分类结果与目标的交叉熵(误差);
    # lasagne.layers.get_all_params(layer):         返回一个列表，包含该层参数的theano 共享变量或表达式 ?
    # bnn_utils.compute_grads(loss, network):       计算梯度;
    # bnn_utilsclipping_scaling(updates, network): 该函数在参数更新后规范化;
    # OrderedDict:                                  有序字典类;
    # dict.items():                                 返回字典的键值;
    # tensor.nep():                                 相当于"a != b";
    # tensor.argmax():                              返回沿指定轴取得最大值的下标;
from __future__ import print_function

import sys
import os
import time
import numpy as np
np.random.seed(1234) # for reproducibility?

# specifying the gpu to use
# import theano.sandbox.cuda
# theano.sandbox.cuda.use('gpu1')
import theano
import theano.tensor as T
import lasagne
import cPickle as pickle
import gzip

```

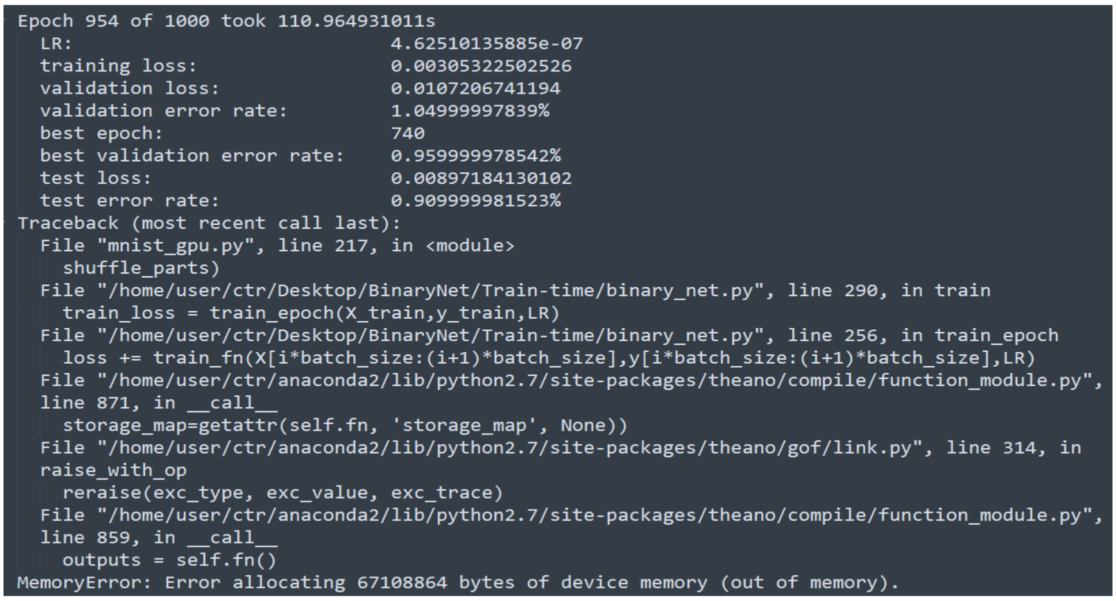
```python
import binary_net

from pylearn2.datasets.zca_dataset import ZCA_Dataset
from pylearn2.datasets.cifar10 import CIFAR10
from pylearn2.utils import serial

from collections import OrderedDict

if __name__ == "__main__":

    # BN parameters (设定超参数)
    batch_size = 50
    print("batch_size = "+str(batch_size))
    # alpha is the exponential moving average factor
    alpha = .1
    print("alpha = "+str(alpha))
    epsilon = 1e-4
    print("epsilon = "+str(epsilon))

    # BinaryOut
    activation = binary_net.binary_tanh_unit
    print("activation = binary_net.binary_tanh_unit")
    # activation = binary_net.binary_sigmoid_unit
    # print("activation = binary_net.binary_sigmoid_unit")

    # BinaryConnect    (设定 BNN-神经网络标识)
    binary = True
    print("binary = "+str(binary))
    stochastic = False
    print("stochastic = "+str(stochastic))
    # (-H,+H) are the two binary values
    # H = "Glorot"
    H = 1.
    print("H = "+str(H))
    # W_LR_scale = 1.
    W_LR_scale = "Glorot" # "Glorot" means we are using the coefficients from Glorot's paper
    print("W_LR_scale = "+str(W_LR_scale))

    # Training parameters
    num_epochs = 500
    print("num_epochs = "+str(num_epochs))

    # Decaying LR   (设定学习速率)
    LR_start = 0.001
    print("LR_start = "+str(LR_start))
    LR_fin = 0.0000003
    print("LR_fin = "+str(LR_fin))
    LR_decay = (LR_fin/LR_start)**(1./num_epochs)
    print("LR_decay = "+str(LR_decay))
    # BTW, LR decay might good for the BN moving average...

    train_set_size = 45000
    print("train_set_size = "+str(train_set_size))
    shuffle_parts = 1
    print("shuffle_parts = "+str(shuffle_parts))

    print('Loading CIFAR-10 dataset...')    (加载数据)

    train_set = CIFAR10(which_set="train",start=0,stop = train_set_size)
    valid_set = CIFAR10(which_set="train",start=train_set_size,stop = 50000)
    test_set = CIFAR10(which_set="test")

    # bc01 format
    # Inputs in the range [-1,+1]
    # print("Inputs in the range [-1,+1]")
    train_set.X = np.reshape(np.subtract(np.multiply(2./255.,train_set.X),1.),(-1,3,32,32))
    valid_set.X = np.reshape(np.subtract(np.multiply(2./255.,valid_set.X),1.),(-1,3,32,32))
    test_set.X = np.reshape(np.subtract(np.multiply(2./255.,test_set.X),1.),(-1,3,32,32))

    # flatten targets
    train_set.y = np.hstack(train_set.y)
    valid_set.y = np.hstack(valid_set.y)
    test_set.y = np.hstack(test_set.y)

    # Onehot the targets
    train_set.y = np.float32(np.eye(10)[train_set.y])
    valid_set.y = np.float32(np.eye(10)[valid_set.y])
    test_set.y = np.float32(np.eye(10)[test_set.y])

    # for hinge loss
    train_set.y = 2* train_set.y - 1.
    valid_set.y = 2* valid_set.y - 1.
    test_set.y = 2* test_set.y - 1.

    print('Building the CNN...')     (构建深度网络)

    # Prepare Theano variables for inputs and targets
    input = T.tensor4('inputs')
    target = T.matrix('targets')
    LR = T.scalar('LR', dtype=theano.config.floatX)

    cnn = lasagne.layers.InputLayer()

    # 128C3-128C3-P2
    cnn = binary_net.Conv2DLayer()
    cnn = lasagne.layers.BatchNormLayer()
    cnn = lasagne.layers.NonlinearityLayer()

    cnn = binary_net.Conv2DLayer()
    cnn = lasagne.layers.MaxPool2DLayer(cnn, pool_size=(2, 2))
    cnn = lasagne.layers.BatchNormLayer()
    cnn = lasagne.layers.NonlinearityLayer()

    # 256C3-256C3-P2
    cnn = binary_net.Conv2DLayer()
    cnn = lasagne.layers.BatchNormLayer()
    cnn = lasagne.layers.NonlinearityLayer()

    cnn = binary_net.Conv2DLayer()
    cnn = lasagne.layers.MaxPool2DLayer(cnn, pool_size=(2, 2))
    cnn = lasagne.layers.BatchNormLayer()
    cnn = lasagne.layers.NonlinearityLayer()

    # 512C3-512C3-P2
    cnn = binary_net.Conv2DLayer()
    cnn = lasagne.layers.BatchNormLayer()
    cnn = lasagne.layers.NonlinearityLayer()

    cnn = binary_net.Conv2DLayer()
    cnn = lasagne.layers.MaxPool2DLayer(cnn, pool_size=(2, 2))
    cnn = lasagne.layers.BatchNormLayer()
    cnn = lasagne.layers.NonlinearityLayer()

    # print(cnn.output_shape)
    # 1024FP-1024FP-10FP
    cnn = binary_net.DenseLayer()
    cnn = lasagne.layers.BatchNormLayer()
    cnn = lasagne.layers.NonlinearityLayer()

    cnn = binary_net.DenseLayer()
    cnn = lasagne.layers.BatchNormLayer()
    cnn = lasagne.layers.NonlinearityLayer()

    cnn = binary_net.DenseLayer()
    cnn = lasagne.layers.BatchNormLayer()
    #(计算网络输出)
    train_output = lasagne.layers.get_output(cnn, deterministic=False)

    # squared hinge loss   (定义误差计算函数)
    loss = T.mean(T.sqr(T.maximum(0.,1.-target*train_output)))

    if binary:

        # W updates   (权重更新)
        W = lasagne.layers.get_all_params(cnn, binary=True)
        W_grads = binary_net.compute_grads(loss,cnn)
        updates = lasagne.updates.adam(loss_or_grads=W_grads, params=W, learning_rate=LR)
        updates = binary_net.clipping_scaling(updates,cnn)

        # other parameters updates    (其它参数更新)
        params = lasagne.layers.get_all_params(cnn, trainable=True, binary=False)
        updates = OrderedDict(updates.items() + lasagne.updates.adam(loss_or_grads=loss, params=params, learning_rate=LR).items())

    else:
        # (参数更新)
        params = lasagne.layers.get_all_params(cnn, trainable=True)
        updates = lasagne.updates.adam(loss_or_grads=loss, params=params, learning_rate=LR)

    # test prediction and loss expressions (测试数据和误差表示)
    test_output = lasagne.layers.get_output(cnn, deterministic=True)
    test_loss = T.mean(T.sqr(T.maximum(0.,1.-target*test_output)))
    test_err = T.mean(T.neq(T.argmax(test_output, axis=1), T.argmax(target, axis=1)),dtype=theano.config.floatX)

    (建立训练数据和测试数据的 theano 函数)
    # Compile a function performing a training step on a mini-batch (by giving the updates dictionary)
```

```python
    # and returning the corresponding training loss:
    train_fn = theano.function([input, target, LR], loss, updates=updates)

    # Compile a second function computing the validation loss and accuracy:
    val_fn = theano.function([input, target], [test_loss, test_err])

    print('Training...')
    binary_net.train(
            train_fn,val_fn,              # 训练和测试数据的 theano 函数
            cnn,                          # 神经网络
            batch_size,                   # 小批量数据大小
            LR_start,LR_decay,            # 学习速率
            num_epochs,                   # 迭代期次数
            train_set.X,train_set.y,      # 训练数据
            valid_set.X,valid_set.y,      # 验证数据
            test_set.X,test_set.y,        # 测试数据
            shuffle_parts=shuffle_parts)
```

2.3 Analysis about Results

But this code need to run two days. So, we can’t write the answer into our report. Fortunately, we also utilize the *mnist* version using binarized neutral network. And it only cost 8 ours to get the parameters. (*mnist* is also a train set which is like *Cifar10.* But it is simpler.)

This is the 954-epoch answer on *mnist* datasets. The k80 GPU memory is too busy。



```
Epoch 954 of 1000 took 110.964931011s
  LR:                           4.62510135885e-07
  training loss:                0.00305322502526
  validation loss:              0.0107206741194
  validation error rate:        1.04999997839%
  best epoch:                   740
  best validation error rate:   0.959999978542%
  test loss:                    0.00897184130102
  test error rate:              0.909999981523%
Traceback (most recent call last):
  File "mnist_gpu.py", line 217, in <module>
    shuffle_parts)
  File "/home/user/ctr/Desktop/BinaryNet/Train-time/binary_net.py", line 290, in train
    train_loss = train_epoch(X_train,y_train,LR)
  File "/home/user/ctr/Desktop/BinaryNet/Train-time/binary_net.py", line 256, in train_epoch
    loss += train_fn(X[i*batch_size:(i+1)*batch_size],y[i*batch_size:(i+1)*batch_size],LR)
  File "/home/user/ctr/anaconda2/lib/python2.7/site-packages/theano/compile/function_module.py",
  line 871, in __call__
    storage_map=getattr(self.fn, 'storage_map', None))
  File "/home/user/ctr/anaconda2/lib/python2.7/site-packages/theano/gof/link.py", line 314, in
  raise_with_op
    reraise(exc_type, exc_value, exc_trace)
  File "/home/user/ctr/anaconda2/lib/python2.7/site-packages/theano/compile/function_module.py",
  line 859, in __call__
    outputs = self.fn()
MemoryError: Error allocating 67108864 bytes of device memory (out of memory).
```

Bibliography

[1] Binarized Neural Networks : Training Neural Networks with Weights and

Activations Constrained to +1 or−1

[2]Bahdanau,Dzmitry,Cho,Kyunghyun,andBengio,Yoshua. Neural machine translation by jointly learning to align and translate. In ICLR’2015, arXiv:1409.0473, 2015.