```
import numpy as np
import time

import os

import chainer
from chainer import cuda, Function, gradient_check, Variable,
optimizers, serializers, utils
from chainer import Link, Chain, ChainList
import chainer.functions as F
import chainer.links as L

#load CIFAR10 data
import h5py
CIFAR10_data = h5py.File('CIFAR10.hdf5', 'r')
x_train = np.float32(CIFAR10_data['X_train'][:] )
y_train = np.int32(np.array(CIFAR10_data['Y_train'][:]))
x_test = np.float32(CIFAR10_data['X_test'][:] )
y_test = np.int32( np.array(CIFAR10_data['Y_test'][:]  ) )

CIFAR10_data.close()


#D = 32
num_outputs = 10

class Conv_NN(Chain):
    def __init__(self):
        super(Conv_NN, self).__init__(
            conv1_1=L.Convolution2D(3, 32, 3, pad=1),
            bn1_1=L.BatchNormalization(32),
            conv1_2=L.Convolution2D(32, 32, 3, pad=1),
            bn1_2=L.BatchNormalization(64),
            conv1_3=L.Convolution2D(32, 32, 3, pad=1),
            bn1_3=L.BatchNormalization(64),
            conv1_4=L.Convolution2D(32, 32, 3, pad=1),
            bn1_4=L.BatchNormalization(32),

            conv2_1=L.Convolution2D(32, 64, 3, pad=1),
            bn2_1=L.BatchNormalization(64),
            conv2_2=L.Convolution2D(64, 64, 3, pad=1),
            bn2_2=L.BatchNormalization(64),

            bn3_1=L.BatchNormalization(128),
            conv3_2=L.Convolution2D(128, 128, 3, pad=1),
            bn3_2=L.BatchNormalization(128),
            conv3_3=L.Convolution2D(128, 128, 3, pad=1),
            bn3_3=L.BatchNormalization(128),
            conv3_4=L.Convolution2D(128, 128, 3, pad=1),
            bn3_3=L.BatchNormalization(128),
```

```
            conv3_5=L.Convolution2D(128, 128, 3, pad=1),
            bn3_5=L.BatchNormalization(128),
            conv3_6=L.Convolution2D(128, 128, 3, pad=1),
            bn3_6=L.BatchNormalization(128),

            fc4 = L.Linear(2048, 500),
            fc5 = L.Linear(500, 500),
            fc6 = L.Linear(500,10),
        )
    def __call__(self, x_data, y_data, dropout_bool, bn_bool, p):
        x = Variable(x_data)
        t = Variable(y_data)
        h = F.relu(self.bn1_1(self.conv1_1(x), bn_bool))
        h = F.relu(self.bn1_2(self.conv1_2(h), bn_bool))
        h = F.relu(self.bn1_3(self.conv1_3(h), bn_bool))
        h = F.relu(self.bn1_4(self.conv1_4(h), bn_bool))
        h = F.max_pooling_2d(h, 3, 2)
        h = F.dropout(h, p, dropout_bool)

        h = F.relu(self.bn2_1(self.conv2_1(h), bn_bool))
        h = F.relu(self.bn2_2(self.conv2_2(h), bn_bool))
        h = F.max_pooling_2d(h, 3, 2)
        h = F.dropout(h, p, dropout_bool)

        h = F.relu(self.bn3_1(self.conv3_1(h), bn_bool))
        h = F.relu(self.bn3_2(self.conv3_2(h), bn_bool))
        h = F.relu(self.bn3_3(self.conv3_3(h), bn_bool))
        h = F.relu(self.bn3_4(self.conv3_4(h), bn_bool))
        h = F.relu(self.bn3_5(self.conv3_5(h), bn_bool))
        h = F.relu(self.bn3_6(self.conv3_6(h), bn_bool))
        h = F.max_pooling_2d(h, 3, 2)
        h = F.dropout(h, p, dropout_bool)

        h = F.dropout(F.relu(self.fc4(h)), p, dropout_bool)
        h = F.dropout(F.relu(self.fc5(h)), p, dropout_bool)
        h = self.fc6(h)
        L_out = h
        return F.softmax_cross_entropy(L_out, t), F.accuracy(L_out, t)


#returns test accuracy of the model.  dropout is set to its test state
def Calculate_Test_Accuracy(x_test, y_test, model, p, GPU_on,
batch_size):
    L_Y_test = len(y_test)
    counter = 0
    test_accuracy_total = 0.0
    for i in range(0, L_Y_test, batch_size):
        if (GPU_on):
            x_batch = cuda.to_gpu(x_test[i:i+ batch_size,:])
            y_batch = cuda.to_gpu(y_test[i:i+ batch_size] )
```

```
        else:
            x_batch = x_test[i:i+batch_size,:]
            y_batch = y_test[i:i+batch_size]
        dropout_bool = False
        bn_bool = True
        loss, accuracy = model(x_batch, y_batch, dropout_bool,bn_bool,
p)
        test_accuracy_batch  = 100.0*np.float(accuracy.data )
        test_accuracy_total += test_accuracy_batch
        counter += 1
    test_accuracy = test_accuracy_total/(np.float(counter))
    return test_accuracy


model =  Conv_NN()

#True if training with GPU, False if training with CPU
GPU_on = True
#GPU_on = False

#size of minibatches
batch_size = 250

#transfer model to GPU
if (GPU_on):
    model.to_gpu()

#optimization method
optimizer = optimizers.MomentumSGD(momentum = .99)
#optimizer = optimizers.RMSprop(lr=0.001, alpha=0.99, eps=1e-08)
optimizer.setup(model)


#learning rate
optimizer.lr = .01

#dropout probability
p = .25

#number of training epochs
num_epochs = 300

L_Y_train = len(y_train)

time1 = time.time()
for epoch in range(num_epochs):
    #reshuffle dataset
    I_permutation = np.random.permutation(L_Y_train)
    x_train = x_train[I_permutation,:]
    y_train = y_train[I_permutation]
```

```
    epoch_accuracy = 0.0
    batch_counter = 0
    for i in range(0, L_Y_train, batch_size):
        if (GPU_on):
            x_batch = cuda.to_gpu(x_train[i:i+batch_size,:])
            y_batch = cuda.to_gpu(y_train[i:i+batch_size] )
        else:
            x_batch = x_train[i:i+batch_size,:]
            y_batch = y_train[i:i+batch_size]
        model.zerograds()
        dropout_bool = True
        bn_bool = False
        loss, accuracy = model(x_batch, y_batch, dropout_bool,
bn_bool, p)
        loss.backward()
        optimizer.update()
        #print("success")
        epoch_accuracy += np.float(accuracy.data)
        batch_counter += 1
    if (epoch % 1 == 0):
        #print "Epoch %d" % epoch
        train_accuracy = 100.0*epoch_accuracy/np.float(batch_counter)
        print "Train accuracy: %f" % train_accuracy
    if (epoch == num_epochs-1):
        test_accuracy = Calculate_Test_Accuracy(x_test, y_test, model,
p, GPU_on, batch_size)
        print "Test Accuracy: %f" % test_accuracy


time2 = time.time()
training_time = time2 - time1
#print "Rank: %d" % rank
print "Training time: %f" % training_time
```