```
import numpy as np
import os
import sys

import time
import math

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

from helperFunctions import getUCF101
from helperFunctions import getVideoFeatures
from helperFunctions import repeatSequence

import h5py

class_list, train, test = getUCF101()

print "LOAD DATA..."
for i in xrange(len(train[0])):
        train[0][i] = train[0][i].split('/')[2].split('.avi')[0]

train_hdf5 = h5py.File('train1024.hdf5','r')
print "Train..."
keys = train_hdf5.keys()
t1 = time.time()
for key in keys:
        data = train_hdf5[key][:]
t2 = time.time()
print t2-t1

for i in xrange(len(test[0])):
        test[0][i] = test[0][i].split('/')[2].split('.avi')[0]

test_hdf5 = h5py.File('test1024.hdf5','r')
print "Test..."
keys = test_hdf5.keys()
t1 = time.time()
for key in keys:
        data = test_hdf5[key][:]
t2 = time.time()
print t2-t1
print "DONE LOADING..."

class RNN(Chain):
    def __init__(self):
        super(RNN, self).__init__(

            lstm_1=L.LSTM(1024,1024),
                bn_1=L.BatchNormalization(1024),

            linear_1=L.Linear(1024,1024),
            bn_linear=L.BatchNormalization(1024),

            out=L.Linear(1024,101)  # the feed-forward output layer
        )

    def reset_state(self):
        self.lstm_1.reset_state()


    def __call__(self, x, t, is_train, bn_bool):

        x = F.dropout(x,train=is_train,ratio=0.5)
        h = self.bn_1(F.dropout(self.lstm_1(x),train=is_train,ratio=0.5),bn_bool)
        h = self.bn_linear(F.dropout(F.sigmoid(self.linear_1(h)),train=is_train,rati
o=0.5),bn_bool)

        h = self.out(h)
        if is_train:
                self.loss = F.softmax_cross_entropy(h,t)
                self.pred = F.softmax(h)
                return self.loss, self.pred
        else:
                self.loss = F.softmax_cross_entropy(h,t)
                self.pred = F.softmax(h)
                return self.pred

rnn = RNN()
# serializers.load_hdf5('RNN.model',rnn)
rnn.to_gpu()
optimizer = optimizers.MomentumSGD(momentum=0.9)
optimizer.setup(rnn)
optimizer.add_hook(chainer.optimizer.GradientClipping(5.0))
optimizer.lr = 0.0005

dataset = train
dataset_length = len(train[0])

batch_size = 384
num_epochs = 25
length_of_sequence = 120

truncated_backprop_length = 30

for epoch in range(num_epochs):

        random_indices = np.random.permutation(dataset_length)

        accuracy = 0.0
        accuracy_counter = 0
        for i in range(0,dataset_length-batch_size,batch_size):

                next_batch = 0

                t1 = time.time()

                hdf5_keys = [train[0][k] for k in random_indices[i:(batch_size+i)]]
                rnn_input_data = []
                count = 0
                for hdf5_key in hdf5_keys:
                        data = train_hdf5[hdf5_key][:]
                        rnn_input_data.append(data)

                for j in xrange(len(rnn_input_data)): # repeat sequence if it's too
short
                        rnn_input_data[j]
                        if(rnn_input_data[j].shape[0]<1.0*length_of_sequence):
                                rnn_input_data[j] = repeatSequence(rnn_input_data[j]
,1.0*length_of_sequence)

                for j in xrange(len(rnn_input_data)): # select random segment of the
 video if it's too long
                        nFrames = rnn_input_data[j].shape[0]
                        if(nFrames>length_of_sequence):
                                start_index = np.random.randint(nFrames-length_of_se
quence)
                                rnn_input_data[j] = rnn_input_data[j][start_index:(s
tart_index+length_of_sequence),:]

                for j in xrange(len(rnn_input_data)): # reverse the sequence
                        if(np.random.randint(2)==1):
                                rnn_input_data[j] = np.flipud(rnn_input_data[j])

                rnn_input_data = np.asarray(rnn_input_data)

                t1_alt = time.time()

                y = train[1][random_indices[i:(batch_size+i)]]
                y = cuda.to_gpu(y.astype(np.int32))
                pred = np.zeros((batch_size,101),np.float32)
                rnn.reset_state()
                rnn.zerograds()
                sequence_length = length_of_sequence-1
                count = 0
                loss_divide = 0.0
                for j in xrange(sequence_length):
                        #scale_factor = 1.0
                        scale_factor = float(j+1)/sequence_length
                        loss_divide += scale_factor

                        x = rnn_input_data[:,j,:]
                        x = cuda.to_gpu(x)

                        curr_loss, current_prediction = rnn(x,y,is_train=True,bn_boo
l=False)
                        if(j==0):
                                loss = scale_factor*curr_loss
                        else:
                                loss += scale_factor*curr_loss

                        count += 1
                        if count % truncated_backprop_length == 0 or count==sequence
_length:
                                rnn.zerograds()
                                loss.backward()
                                loss.unchain_backward()
                                optimizer.update()

                        current_prediction.to_cpu()
                        pred[:,:] += scale_factor*np.log(current_prediction.data)

                t2 = time.time()

                y = train[1][random_indices[i:(batch_size+i)]]

                batch_accuracy = 0.0
                predictions = np.argmax(pred,axis=1)
                for k in xrange(y.size):
                        if(y[k]==predictions[k]):
                                accuracy += 1.0
                                batch_accuracy += 1.0
                accuracy_counter += batch_size

                print epoch, i, "%.2f" % float(t2-t1), "%.2f" % float(t2-t1_alt), "B
atchLoss: %.3f" % float(loss.data/loss_divide), "BatchAccuracy: %.3f" % float(batch_
accuracy/batch_size), " Accuracy: %.3f" % float(accuracy/accuracy_counter), length_o
f_sequence

serializers.save_hdf5('RNN.model',rnn)



print "TEST SET"
test_length = len(test[0])
random_indices = np.random.permutation(test_length)

batch_size = 384/4
length_of_sequence = 180
accuracy = 0.0
accuracy_counter = 0
accuracy_top5 = 0.0
accuracy_top10 = 0.0
test_runs = 1

for i in range(0,test_length-batch_size,batch_size):

        next_batch = 0

        t1 = time.time()

        y = test[1][random_indices[i:(batch_size+i)]]
        y = cuda.to_gpu(y.astype(np.int32))
        hdf5_keys = [test[0][k] for k in random_indices[i:(batch_size+i)]]
        rnn_input_data = []
        count = 0
        for hdf5_key in hdf5_keys:
                data = test_hdf5[hdf5_key][:]
                rnn_input_data.append(data)
        rnn_input_data_raw = rnn_input_data


        pred = np.zeros((batch_size,101),np.float32)

        for test_run in xrange(test_runs):
                rnn_input_data = rnn_input_data_raw
                for j in xrange(len(rnn_input_data)):
                        if(rnn_input_data[j].shape[0]<1.0*length_of_sequence):
                                rnn_input_data[j] = repeatSequence(rnn_input_data[j]
,1.0*length_of_sequence)

                for j in xrange(len(rnn_input_data)):
                        nFrames = rnn_input_data[j].shape[0]
                        if(nFrames>length_of_sequence):
                                start_index = np.random.randint(nFrames-length_of_se
quence)
                                rnn_input_data[j] = rnn_input_data[j][start_index:(s
tart_index+length_of_sequence),:]

                rnn_input_data_np = np.asarray(rnn_input_data)

                rnn.reset_state()
                rnn.zerograds()
                sequence_length = float(length_of_sequence-1)
                for j in xrange(length_of_sequence-1):
                        scale_factor = float(j+1)/length_of_sequence # 1.0

                        x = rnn_input_data_np[:,j,:]
                        x = cuda.to_gpu(x)

                        current_prediction = rnn(x,y,is_train=False,bn_bool=True)

                        current_prediction.to_cpu()

                        pred[:,:] += scale_factor*np.log(current_prediction.data)


        t2 = time.time()

        y = test[1][random_indices[i:(batch_size+i)]]

        argsort_pred = np.argsort(-pred)
        argsort_pred = argsort_pred[:,0:10]
        predictions = np.argmax(pred,axis=1)
        for k in xrange(y.size):
                if(y[k]==predictions[k]):
                        accuracy += 1.0
                if(np.any(argsort_pred[k,:]==y[k])):
                        accuracy_top10 += 1.0
                if(np.any(argsort_pred[k,0:5]==y[k])):
                        accuracy_top5 += 1.0

        accuracy_counter += batch_size

        print i, "%.2f" % float(t2-t1), " Accuracy: (%.3f,%.3f,%.3f)" % (float(accur
acy/accuracy_counter),float(accuracy_top5/accuracy_counter),float(accuracy_top10/acc
uracy_counter)), length_of_sequence
```