```
#-*- coding: utf-8 -*-
#__author__ = "Xinpeng.Chen"
_author__ = 'tylin'
from tokenizer.ptbtokenizer import PTBTokenizer
import tensorflow as tf
import pandas as pd
import numpy as np
import os
import sys
#import ipdb
import time
import cv2
import scipy.io
from keras.preprocessing import sequence
import matplotlib.pyplot as plt
from cider.cider import Cider
from rouge.rouge import Rouge
from bleu.bleu import Bleu
from tensorflow.python.ops import tensor_array_ops, control_flow_ops
from tensorflow.python.layers import core as layers_core

from process_word import load_bin_vec, add_unknown_words, get_W
import dynamic
import basic_decoder

tf.set_random_seed(20180525)


class Video_Caption_Generator():
    def __init__(self, dim_image, n_words, dim_hidden, batch_size, n_lstm_steps, n_video_lstm_step, n_caption_lstm_step, beam_width, drop_prob,kernel,n_filters, dim_video, n_3d_lstm_step,bias_init_vector=None):
        self.dim_image = dim_image
        self.dim_video = dim_video
        self.n_words = n_words
        self.dim_hidden = dim_hidden
        self.batch_size = batch_size
        self.n_lstm_steps = n_lstm_steps
        self.n_video_lstm_step = n_video_lstm_step
        self.n_3d_lstm_step = n_3d_lstm_step
        self.n_caption_lstm_step = n_caption_lstm_step
        self.beam_width = beam_width
        self.drop_prob = drop_prob
        self.kernel = kernel
```

```
        self.n_filters = n_filters

        with tf.device("/gpu:0"):
            self.Wemb = tf.Variable(tf.random_uniform([n_words, 300], -0.05, 0.05), name='Wemb')

        self.lstm1 = tf.nn.rnn_cell.BasicLSTMCell(dim_hidden, state_is_tuple=True)
        self.lstm2 = tf.nn.rnn_cell.BasicLSTMCell(dim_hidden, state_is_tuple=True)
        self.lstm4 = tf.nn.rnn_cell.BasicLSTMCell(dim_hidden, state_is_tuple=True)
        self.lstm5 = tf.nn.rnn_cell.BasicLSTMCell(dim_hidden, state_is_tuple=True)
        #self.lstm3 = tf.nn.rnn_cell.BasicLSTMCell(dim_hidden, state_is_tuple=True)
        #self.lstm1 = tf.contrib.rnn.LayerNormBasicLSTMCell(dim_hidden)
        #self.lstm2 = tf.contrib.rnn.LayerNormBasicLSTMCell(dim_hidden)
        self.lstm3 = tf.contrib.rnn.LayerNormBasicLSTMCell(dim_hidden)



        self.encode_image_W = tf.Variable( tf.random_uniform([12844, dim_hidden], -0.05, 0.05), name='encode_image_W')
        self.encode_image_b = tf.Variable( tf.zeros([dim_hidden]), name='encode_image_b')

        self.encode_video_W = tf.Variable( tf.random_uniform([dim_video, dim_hidden], -0.05, 0.05), name='encode_video_W')
        self.encode_video_b = tf.Variable( tf.zeros([dim_hidden]), name='encode_video_b')

        self.encode_lstm_hW = tf.Variable( tf.random_uniform([12844, dim_hidden], -0.05, 0.05), name='encode_lstm_hW')
        self.encode_lstm_hb = tf.Variable( tf.zeros([dim_hidden]), name='encode_lstm_hb')

        self.encode_lstm_cW = tf.Variable( tf.random_uniform([12844, dim_hidden], -0.05, 0.05), name='encode_lstm_cW')
        self.encode_lstm_cb = tf.Variable( tf.zeros([dim_hidden]), name='encode_lstm_cb')

        self.vse_W = tf.Variable(tf.random_uniform([300, 512], -0.05, 0.05), name='vse_W')
        self.sim_W = tf.Variable(tf.random_uniform([1024,512], -0.05, 0.05), name='sim_W')



        self.embed_word_W = tf.Variable(tf.random_uniform([dim_hidden, n_words], -0.05,0.05), name='embed_word_W')
        if bias_init_vector is not None:
            self.embed_word_b = tf.Variable(bias_init_vector.astype(np.float32), name='embed_word_b')
        else:
```

```
            self.embed_word_b = tf.Variable(tf.zeros([n_words]), name='embed_word_b')

    def build_model(self):
        video = tf.placeholder(tf.float32, [self.batch_size, self.n_video_lstm_step, self.dim_image])
        video_mask = tf.placeholder(tf.float32, [self.batch_size, self.n_video_lstm_step])

        #sim_feature = tf.placeholder(tf.float32, [self.batch_size, self.n_caption_lstm_step, 1024])
        #sim_caption = tf.placeholder(tf.float32, [self.batch_size, self.n_caption_lstm_step+1])


        embed_placeholder = tf.placeholder(tf.float32, [self.n_words, 300])

        embed_init = self.Wemb.assign(embed_placeholder)
        #concate = tf.placeholder(tf.float32, [self.batch_size, 300])
        audio = tf.placeholder(tf.float32, [self.batch_size, self.n_video_lstm_step, 9984])
        cate = tf.placeholder(tf.float32, [self.batch_size, self.n_video_lstm_step, 300])

        #video_audio = tf.concat([video, audio], 2)
        video_audio = tf.concat([video, audio, cate], 2)
        lstm_input = tf.reduce_mean(video_audio, 1)

        lstm_h = tf.nn.xw_plus_b(lstm_input, self.encode_lstm_hW, self.encode_lstm_hb)
        lstm_c = tf.nn.xw_plus_b(lstm_input, self.encode_lstm_cW, self.encode_lstm_cb)

        video_3d = tf.placeholder(tf.float32, [self.batch_size, self.n_3d_lstm_step, self.dim_video])

        reward = tf.placeholder(tf.float32, [self.batch_size, self.n_caption_lstm_step])
        prob = tf.placeholder(tf.float32)

        flag = tf.placeholder(tf.bool)

        caption = tf.placeholder(tf.int32, [self.batch_size, self.n_caption_lstm_step+1])
        caption_mask = tf.placeholder(tf.float32, [self.batch_size, self.n_caption_lstm_step+1])

        video_flat = tf.reshape(video_audio, [-1, 12844])          #gaibian shape shizhi chengfa chengli
        image_emb = tf.nn.xw_plus_b( video_flat, self.encode_image_W, self.encode_image_b ) # (batch_size*n_lstm_steps, dim_hidden)
        image_emb = tf.reshape(image_emb, [self.batch_size, self.n_lstm_steps, self.dim_hidden])
```

```
        video_3d_flat = tf.reshape(video_3d, [-1, self.dim_video])          #gaibian shape shizhi chengfa  chengli
        video_emb = tf.nn.xw_plus_b( video_3d_flat, self.encode_video_W, self.encode_video_b ) # (batch_size*n_lstm_steps, dim_hidden)
        video_emb = tf.reshape(video_emb, [self.batch_size, self.n_3d_lstm_step, self.dim_hidden])

        image_emb1 = tf.expand_dims(image_emb, -1)
        F1 = tf.Variable(tf.random_normal([self.kernel, self.dim_hidden ,1, self.n_filters[0]] ,-0.05, 0.05),name = 'F1')
        FB1 = tf.Variable(tf.constant(0.1, shape=[self.n_filters[0]]),name = 'FB1')
        C1 = tf.nn.relu(tf.add(tf.nn.conv2d(image_emb1, F1, [1,1,1,1], padding='VALID'), FB1))
        C1 = tf.nn.max_pool(C1, [1,2,1,1] , [1,2,1,1], padding='VALID')
        C1 = tf.expand_dims(tf.squeeze(C1), -1)

        F2 = tf.Variable(tf.random_normal([self.kernel, self.n_filters[0] ,1, self.n_filters[1]] ,-0.05, 0.05),name = 'F2')
        FB2 = tf.Variable(tf.constant(0.1, shape=[self.n_filters[1]]),name = 'FB2')
        C2 = tf.nn.relu(tf.add(tf.nn.conv2d(C1, F2, [1,1,1,1], padding='VALID'), FB2))
        C2 = tf.nn.max_pool(C2, [1,2,1,1] , [1,2,1,1], padding='VALID')
        C2 = tf.expand_dims(tf.squeeze(C2), -1)

        F3 = tf.Variable(tf.random_normal([self.kernel, self.n_filters[1] ,1, self.n_filters[2]] ,-0.05, 0.05),name = 'F3')
        FB3 = tf.Variable(tf.constant(0.1, shape=[self.n_filters[2]]),name = 'FB3')
        C3 = tf.nn.relu(tf.add(tf.nn.conv2d(C2, F3, [1,1,1,1], padding='VALID'), FB3))
        C3 = tf.nn.max_pool(C3, [1,2,1,1] , [1,2,1,1], padding='VALID')
        C3 = tf.expand_dims(tf.squeeze(C3), -1)

        F4 = tf.Variable(tf.random_normal([self.kernel, self.n_filters[2] ,1, self.n_filters[3]] ,-0.05, 0.05),name = 'F4')
        FB4 = tf.Variable(tf.constant(0.1, shape=[self.n_filters[3]]),name = 'FB4')
        C4 = tf.nn.relu(tf.add(tf.nn.conv2d(C3, F4, [1,1,1,1], padding='VALID'), FB4))
        C4 = tf.nn.max_pool(C4, [1,C4.get_shape()[1],1,1] , [1,1,1,1], padding='VALID')
        C4 = tf.squeeze(C4)


        state1 = self.lstm1.zero_state(batch_size=self.batch_size, dtype=tf.float32)
        state2 = self.lstm2.zero_state(batch_size=self.batch_size, dtype=tf.float32)

        padding = tf.zeros([self.batch_size, self.dim_hidden])

        probs = []
        loss = 0.0
```

```
        #loss2 = 0.0

        decoder_outputs = []
        current_embeds = []
        current_words = []

        self.lstm1 = tf.contrib.rnn.DropoutWrapper(self.lstm1, output_keep_prob = 0.5)
        self.lstm2 = tf.contrib.rnn.DropoutWrapper(self.lstm2, output_keep_prob = 0.5)
        self.lstm4 = tf.contrib.rnn.DropoutWrapper(self.lstm4, output_keep_prob = 0.5)
        self.lstm5 = tf.contrib.rnn.DropoutWrapper(self.lstm5, output_keep_prob = 0.5)

        with tf.variable_scope("bilstm1"):
            bi_outputs, (encoder_state_fw, encoder_state_bw) = tf.nn.bidirectional_dynamic_rnn(self.lstm1,
                                                                                               self.lstm2,
                                                                                               image_emb,
                                                                                               dtype = tf.float32,
                                                                                               time_major=False)



        encoder_outputs = tf.concat(bi_outputs, -1)
        #encoder_state = tf.concat(encoder_state, -1)

        #encoder_state_c = (encoder_state_fw.c + encoder_state_bw.c)/2
        encoder_state_h = (encoder_state_fw.h + encoder_state_bw.h)/2
        #encoder_state = tf.nn.rnn_cell.LSTMStateTuple(encoder_state_c, encoder_state_h)
        encoder_state = tf.nn.rnn_cell.LSTMStateTuple(lstm_c, lstm_h)

        with tf.variable_scope("bilstm2"):
            bi_outputs_video, (encoder_state_fw_video, encoder_state_bw_video) = tf.nn.bidirectional_dynamic_rnn(self.lstm4,
                                                                                               self.lstm5,
                                                                                               video_emb,
                                                                                               dtype = tf.float32,
                                                                                               time_major=False)



        encoder_outputs_video = tf.concat(bi_outputs_video, -1)
        #encoder_state = tf.concat(encoder_state, -1)

        #encoder_state_c = (encoder_state_fw.c + encoder_state_bw.c)/2
        #encoder_state_h = (encoder_state_fw.h + encoder_state_bw.h)/2
```

```
        #encoder_state = tf.nn.rnn_cell.LSTMStateTuple(encoder_state_c, encoder_state_h)

        attention_states = tf.transpose(encoder_outputs, [0, 1, 2])
        attention_states_video = tf.transpose(encoder_outputs_video, [0,1,2])

        #attention_states = tf.contrib.seq2seq.tile_batch(attention_states, multiplier=beam_width)

        for i in range(0,self.n_caption_lstm_step):
            current_embed = tf.nn.embedding_lookup(self.Wemb, caption[:,i])
            current_word = current_embed

            current_embed = tf.concat([current_embed, C4],1)
            labels = tf.expand_dims(caption[:,i+1], 1)
            indices = tf.expand_dims(tf.range(0, self.batch_size, 1), 1)
            concated = tf.concat([indices, labels], 1)
            onehot_labels = tf.sparse_to_dense(concated, tf.stack([self.batch_size, self.n_words]), 1.0, 0.0)
            decoder_outputs.append(onehot_labels)
            current_embeds.append(current_embed)
            current_words.append(current_word)

        current_embeds = tf.transpose(current_embeds, [1, 0, 2])
        decoder_outputs = tf.transpose(decoder_outputs, [1, 0, 2])
        current_words = tf.transpose(current_words, [1,0,2])

        sim_loss = 0


        vse = tf.multiply(current_words[:,1:,:], tf.expand_dims(caption_mask[:,2:self.n_caption_lstm_step+1], 2))
        vse = tf.reduce_sum(vse, 1)
        caption_vse = tf.expand_dims(tf.reduce_sum(caption_mask[:,2:self.n_caption_lstm_step+1],1), 1)
        vse = tf.div(vse, caption_vse)

        vse = tf.matmul(vse, self.vse_W)        #vse and C4 shape[512,512]


        #gen_sim = tf.multiply(sim_feature[:,:self.n_caption_lstm_step-1,:], tf.expand_dims(sim_caption[:,2:self.n_caption_lstm_step+1], 2))

        #gen_sim = tf.reduce_sum(gen_sim, 1)
        #caption_gen =
```

```
tf.expand_dims(tf.reduce_sum(sim_caption[:,2:self.n_caption_lstm_step+1],1), 1)
        #gen_sim = tf.div(gen_sim, caption_gen)
        #gen_sim = tf.matmul(gen_sim, self.sim_W)


        C4 = tf.reduce_mean(image_emb, 1)
        C4_distance = tf.expand_dims(tf.sqrt(tf.reduce_mean(tf.square(C4),1)), 1)

        vse_distance = tf.expand_dims(tf.sqrt(tf.reduce_mean(tf.square(vse),1)), 0)

        sim_matrix = tf.matmul(C4, tf.transpose(vse, [1,0]))
        sim_distance = tf.matmul(C4_distance, vse_distance) + tf.constant(0.00001)
        sim_matrix = tf.div(sim_matrix, sim_distance)

        '''
        #sim_loss = 0
        for i in range(self.batch_size):                                #video retrieve loss
            sim_row = sim_matrix[i,:]
            sim_column = sim_matrix[:,i]

            sim_index = sim_matrix[i,i]

            index = [j for j in range(batch_size) if j!=i]
            sim_row = tf.gather(sim_row, index)
            sim_column = tf.gather(sim_column, index)

            #sim_row_max = tf.reduce_mean(sim_row)
            #sim_column_max = tf.reduce_mean(sim_column)

            sim_row = sim_row - sim_index
            sim_column = sim_column - sim_index

            sim_row_max = tf.reduce_mean(tf.nn.relu(0.2 + sim_row))
            sim_column_max = tf.reduce_mean(tf.nn.relu(0.2 + sim_column))

            #sim_row = tf.expand_dims(tf.squeeze(sim_row),0)
            #sim_column = tf.expand_dims(tf.squeeze(sim_column),0)
            #sim_row = tf.expand_dims(sim_row, -1)
            #sim_column = tf.expand_dims(sim_column, -1)
            #sim_row = tf.expand_dims(sim_row, -1)
            #sim_column = tf.expand_dims(sim_column, -1)

            #sim_row_max = tf.nn.max_pool(sim_row, [1,self.batch_size-1,1,1], [1,2,1,1],
'VALID')
```

```
                #sim_column_max = tf.nn.max_pool(sim_column,
[1,self.batch_size-1,1,1],[1,2,1,1], 'VALID')


                sim_loss= sim_loss + sim_row_max + sim_column_max
            '''

            sim_loss = tf.reduce_sum(tf.sqrt(tf.reduce_mean(tf.square(C4-vse),1)))


            sim_loss = sim_loss/self.batch_size


            helper = tf.contrib.seq2seq.TrainingHelper(current_embeds,
                                                       sequence_length=tf.fill([self.batch_size],
self.n_caption_lstm_step),
                                                       time_major=False)
            def embed_and_input_proj(inputs):
                a = tf.nn.embedding_lookup(self.Wemb, inputs)
                return tf.concat([a, C4], 1)
            start = tf.fill([self.batch_size], tf.constant(1,dtype = tf.int32))
            end = tf.constant(20000,dtype = tf.int32)

            helper2 = tf.contrib.seq2seq.GreedyEmbeddingHelper(start_tokens=start,
                                                               end_token=end,
                                                               embedding=embed_and_input_proj)



            with tf.variable_scope("ATT"):
                attention_mechanism =
tf.contrib.seq2seq.BahdanauAttention(dim_hidden,attention_states)
                attention_mechanism_video =
tf.contrib.seq2seq.BahdanauAttention(dim_hidden,video_3d)
                decoder = tf.contrib.seq2seq.AttentionWrapper(self.lstm3,
[attention_mechanism,attention_mechanism_video],
attention_layer_size=[dim_hidden,dim_hidden])
                #decoder = tf.contrib.seq2seq.AttentionWrapper(self.lstm3,
attention_mechanism_video, attention_layer_size=dim_hidden)

                state3 = decoder.zero_state(batch_size=self.batch_size, dtype=tf.float32)
                state3 = state3.clone(cell_state=encoder_state)

                #decoder = tf.contrib.seq2seq.AttentionWrapper(decoder,
```

```
attention_mechanism_video, attention_layer_size=dim_hidden)
                #decoder = tf.contrib.seq2seq.AttentionWrapper(decoder, attention_mechanism,
attention_layer_size=dim_hidden)
            #state3 = self.lstm3.zero_state(batch_size=self.batch_size, dtype=tf.float32)
                projection_layer = layers_core.Dense(self.n_words, use_bias=False)

                #projection_layer2 = layers_core.Dense(self.n_words, use_bias=False)
                #state3 = decoder.zero_state(batch_size=self.batch_size, dtype=tf.float32)
                #state3 = state3.clone(cell_state=encoder_state)
                #state3 = state3.clone(cell_state=state3)
            #state3 = decoder.zero_state(batch_size=self.batch_size * beam_width,
dtype=tf.float32)

                decoder1 = tf.contrib.rnn.DropoutWrapper(decoder, output_keep_prob = prob)
                decoder2 = decoder

                decoder_cell1 = basic_decoder.BasicDecoder(decoder1, helper, state3,
output_layer=projection_layer)
                decoder_cell2 = basic_decoder.BasicDecoder(decoder2, helper2, state3,
output_layer=projection_layer)



            #decoder = tf.contrib.seq2seq.AttentionWrapper(self.lstm3, attention_mechanism,
attention_layer_size=dim_hidden)

                outputs, _, _,_ = dynamic.dynamic_decode(decoder_cell1,
                        output_time_major=False,
                        impute_finished=True,
                        maximum_iterations=self.n_caption_lstm_step
                                                                )

                outputs2, _, _,hidden_out = dynamic.dynamic_decode(decoder_cell2,
                        output_time_major=False,
                        #impute_finished=True,
                        maximum_iterations=n_caption_lstm_step)

            logits = outputs.rnn_output

            crossent =
tf.nn.sparse_softmax_cross_entropy_with_logits(labels=caption[:,1:self.n_caption_lstm_step + 1],
logits=logits)

            train_loss = (tf.reduce_sum(crossent * caption_mask[:,1:self.n_caption_lstm_step + 1])
```

```
/ batch_size)
            loss = train_loss + 0.1*sim_loss

            senten = outputs2.sample_id
            #loss = train_loss




            return loss, video, video_mask, caption, caption_mask, reward, flag, prob, video_3d, audio, cate, embed_placeholder, embed_init,senten



        def build_generator(self):
            video = tf.placeholder(tf.float32, [self.batch_size, self.n_video_lstm_step, self.dim_image])
            video_mask = tf.placeholder(tf.float32, [self.batch_size, self.n_video_lstm_step])
            video_3d = tf.placeholder(tf.float32, [self.batch_size, self.n_3d_lstm_step, self.dim_video])

            audio = tf.placeholder(tf.float32, [self.batch_size, self.n_video_lstm_step, 9984])
            cate = tf.placeholder(tf.float32, [self.batch_size, self.n_video_lstm_step, 300])

            #video_audio = tf.concat([video, audio], 2)

            video_audio = tf.concat([video, audio, cate], 2)
            lstm_input = tf.reduce_mean(video_audio, 1)

            lstm_h = tf.nn.xw_plus_b(lstm_input, self.encode_lstm_hW, self.encode_lstm_hb)
            lstm_c = tf.nn.xw_plus_b(lstm_input, self.encode_lstm_cW, self.encode_lstm_cb)

            video_flat = tf.reshape(video_audio, [-1, 12844])
            image_emb = tf.nn.xw_plus_b(video_flat, self.encode_image_W, self.encode_image_b)
            image_emb = tf.reshape(image_emb, [self.batch_size, self.n_video_lstm_step, self.dim_hidden])

            video_3d_flat = tf.reshape(video_3d, [-1, self.dim_video])          #gaibian shape shizhi chengfa  chengli
            video_emb = tf.nn.xw_plus_b( video_3d_flat, self.encode_video_W, self.encode_video_b ) # (batch_size*n_lstm_steps, dim_hidden)
            video_emb = tf.reshape(video_emb, [self.batch_size, self.n_3d_lstm_step, self.dim_hidden])
```

```
        image_emb1 = tf.expand_dims(image_emb, -1)
        F1 = tf.Variable(tf.random_normal([self.kernel, self.dim_hidden ,1, self.n_filters[0]] ,-0.05, 0.05),name = 'F1')
        FB1 = tf.Variable(tf.constant(0.1, shape=[self.n_filters[0]]),name = 'FB1')
        C1 = tf.nn.relu(tf.add(tf.nn.conv2d(image_emb1, F1, [1,1,1,1], padding='VALID'), FB1))
        C1 = tf.nn.max_pool(C1, [1,2,1,1] , [1,2,1,1], padding='VALID')
        C1 = tf.expand_dims(tf.squeeze(C1), -1)

        F2 = tf.Variable(tf.random_normal([self.kernel, self.n_filters[0] ,1, self.n_filters[1]] ,-0.05, 0.05),name = 'F2')
        FB2 = tf.Variable(tf.constant(0.1, shape=[self.n_filters[1]]),name = 'FB2')
        C2 = tf.nn.relu(tf.add(tf.nn.conv2d(C1, F2, [1,1,1,1], padding='VALID'), FB2))
        C2 = tf.nn.max_pool(C2, [1,2,1,1] , [1,2,1,1], padding='VALID')
        C2 = tf.expand_dims(tf.squeeze(C2), -1)

        F3 = tf.Variable(tf.random_normal([self.kernel, self.n_filters[1] ,1, self.n_filters[2]] ,-0.05, 0.05),name = 'F3')
        FB3 = tf.Variable(tf.constant(0.1, shape=[self.n_filters[2]]),name = 'FB3')
        C3 = tf.nn.relu(tf.add(tf.nn.conv2d(C2, F3, [1,1,1,1], padding='VALID'), FB3))
        C3 = tf.nn.max_pool(C3, [1,2,1,1] , [1,2,1,1], padding='VALID')
        C3 = tf.expand_dims(tf.squeeze(C3), -1)

        F4 = tf.Variable(tf.random_normal([self.kernel, self.n_filters[2] ,1, self.n_filters[3]] ,-0.05, 0.05),name = 'F4')
        FB4 = tf.Variable(tf.constant(0.1, shape=[self.n_filters[3]]),name = 'FB4')
        C4 = tf.nn.relu(tf.add(tf.nn.conv2d(C3, F4, [1,1,1,1], padding='VALID'), FB4))
        C4 = tf.nn.max_pool(C4, [1,C4.get_shape()[1],1,1] , [1,1,1,1], padding='VALID')
        C4 = tf.squeeze(C4)

        #self.lstm1 = tf.contrib.rnn.DropoutWrapper(self.lstm1, output_keep_prob = prob)
        #self.lstm2 = tf.contrib.rnn.DropoutWrapper(self.lstm2, output_keep_prob = prob)

        #concate = tf.placeholder(tf.float32, [self.batch_size, 300])

        generated_words = []

        probs = []
        embeds = []

        with tf.variable_scope("bilstm1"):
            bi_outputs, (encoder_state_fw, encoder_state_bw) = tf.nn.bidirectional_dynamic_rnn(self.lstm1,
                                                                                               self.lstm2,
```

```
                                                    image_emb,

                                                    dtype = tf.float32,
                                                    time_major=False)

        encoder_outputs = tf.concat(bi_outputs, -1)
        #encoder_state = tf.concat(encoder_state, -1)
        #encoder_state_c = (encoder_state_fw.c + encoder_state_bw.c)/2
        #encoder_state_h = (encoder_state_fw.h + encoder_state_bw.h)/2
        #encoder_state = tf.nn.rnn_cell.LSTMStateTuple(encoder_state_c, encoder_state_h)
        encoder_state = tf.nn.rnn_cell.LSTMStateTuple(lstm_c, lstm_h)

        with tf.variable_scope("bilstm2"):
            bi_outputs_video, (encoder_state_fw_video, encoder_state_bw_video) =
tf.nn.bidirectional_dynamic_rnn(self.lstm4,
                                                    self.lstm5,
                                                    video_emb,
                                                    dtype = tf.float32,
                                                    time_major=False)



        encoder_outputs_video = tf.concat(bi_outputs_video, -1)


        def embed_and_input_proj(inputs):
            a = tf.nn.embedding_lookup(self.Wemb, inputs)
            return tf.concat([a, C4], 1)

        attention_states = tf.transpose(encoder_outputs, [0, 1, 2])
        attention_states_video = tf.transpose(encoder_outputs_video, [0,1,2])
        #attention_states = tf.contrib.seq2seq.tile_batch(attention_states,
multiplier=beam_width)

        start = tf.fill([self.batch_size], tf.constant(1,dtype = tf.int32))
        end = tf.constant(20000,dtype = tf.int32)

        helper = tf.contrib.seq2seq.GreedyEmbeddingHelper(start_tokens=start,
                                                    end_token=end,
                                                    embedding=embed_and_input_proj)
        with tf.variable_scope("ATT"):
            attention_mechanism =
tf.contrib.seq2seq.BahdanauAttention(dim_hidden,attention_states)
            attention_mechanism_video =
```

```
tf.contrib.seq2seq.BahdanauAttention(dim_hidden,video_3d)
            decoder = tf.contrib.seq2seq.AttentionWrapper(self.lstm3,
[attention_mechanism,attention_mechanism_video],
attention_layer_size=[dim_hidden,dim_hidden])
            #decoder = tf.contrib.seq2seq.AttentionWrapper(self.lstm3,
attention_mechanism_video, attention_layer_size=dim_hidden)

            state3 = decoder.zero_state(batch_size=self.batch_size, dtype=tf.float32)
            state3 = state3.clone(cell_state=encoder_state)

            #decoder = tf.contrib.seq2seq.AttentionWrapper(decoder,
attention_mechanism_video, attention_layer_size=dim_hidden)
            #decoder = tf.contrib.seq2seq.AttentionWrapper(decoder, attention_mechanism,
attention_layer_size=dim_hidden)
        #state3 = lstm3.zero_state(self.batch_size)

            projection_layer = layers_core.Dense(self.n_words, use_bias=False)



            #state3 = decoder.zero_state(batch_size=self.batch_size, dtype=tf.float32)
            #state3 = state3.clone(cell_state=encoder_state)
            #state3 = state3.clone(cell_state=state3)
        #state3 = decoder.zero_state(batch_size=beam_width, dtype=tf.float32)

        #decoder = tf.contrib.rnn.DropoutWrapper(decoder, input_keep_prob = 0.5,
output_keep_prob = 0.5)
            decoder_cell = basic_decoder.BasicDecoder(decoder, helper,
state3 ,output_layer=projection_layer)

        #decoder_cell = tf.contrib.seq2seq.BeamSearchDecoder(decoder,
        #                                    embedding=embed_and_input_proj,
        #                                    start_tokens=start,
        #                                    end_token=end,
        #                                    initial_state=state3,
        #                                    beam_width = beam_width,
        #                                    output_layer=projection_layer)

        #decoder = tf.contrib.seq2seq.AttentionWrapper(self.lstm3, attention_mechanism,
attention_layer_size=dim_hidden)
            outputs, _, _,hidden_out = dynamic.dynamic_decode(decoder_cell,
                    output_time_major=False,
                    #impute_finished=True,
                    maximum_iterations=n_caption_lstm_step)
```

```
        generated_words = outputs.sample_id
        #generated_words = tf.squeeze(tf.multinomial(outputs.rnn_output, -1))




        return video, video_mask, generated_words, video_3d, audio, cate, hidden_out



    def build_generator_single(self):
        video = tf.placeholder(tf.float32, [1, self.n_video_lstm_step, self.dim_image])
        video_mask = tf.placeholder(tf.float32, [1, self.n_video_lstm_step])
        video_3d = tf.placeholder(tf.float32, [1, self.n_3d_lstm_step, self.dim_video])

        audio = tf.placeholder(tf.float32, [1, self.n_video_lstm_step, 9984])
        cate = tf.placeholder(tf.float32, [1, self.n_video_lstm_step, 300])

        #video_audio = tf.concat([video, audio], 2)
        video_audio = tf.concat([video, audio, cate], 2)
        lstm_input = tf.reduce_mean(video_audio, 1)

        lstm_h = tf.nn.xw_plus_b(lstm_input, self.encode_lstm_hW, self.encode_lstm_hb)
        lstm_c = tf.nn.xw_plus_b(lstm_input, self.encode_lstm_cW, self.encode_lstm_cb)

        video_flat = tf.reshape(video_audio, [-1, 12844])
        image_emb = tf.nn.xw_plus_b(video_flat, self.encode_image_W, self.encode_image_b)
        image_emb = tf.reshape(image_emb, [1, self.n_video_lstm_step, self.dim_hidden])
        #concate = tf.placeholder(tf.float32, [1, self.beam_width, 300])

        video_3d_flat = tf.reshape(video_3d, [-1, self.dim_video])          #gaibian shape shizhi chengfa  chengli
        video_emb = tf.nn.xw_plus_b( video_3d_flat, self.encode_video_W, self.encode_video_b ) # (batch_size*n_lstm_steps, dim_hidden)
        video_emb = tf.reshape(video_emb, [1, self.n_3d_lstm_step, self.dim_hidden])

        image_emb1 = tf.expand_dims(image_emb, -1)
        F1 = tf.Variable(tf.random_normal([self.kernel, self.dim_hidden ,1, self.n_filters[0]] ,-0.05, 0.05),name = 'F1')
        FB1 = tf.Variable(tf.constant(0.1, shape=[self.n_filters[0]]),name = 'FB1')
        C1 = tf.nn.relu(tf.add(tf.nn.conv2d(image_emb1, F1, [1,1,1,1], padding='VALID'), FB1))
        C1 = tf.nn.max_pool(C1, [1,2,1,1] , [1,2,1,1], padding='VALID')
        C1 = tf.expand_dims(tf.squeeze(C1,[2]), -1)
```

```
        F2 = tf.Variable(tf.random_normal([self.kernel, self.n_filters[0] ,1, self.n_filters[1]] ,-0.05,
0.05),name = 'F2')
        FB2 = tf.Variable(tf.constant(0.1, shape=[self.n_filters[1]]),name = 'FB2')
        C2 = tf.nn.relu(tf.add(tf.nn.conv2d(C1, F2, [1,1,1,1], padding='VALID'), FB2))
        C2 = tf.nn.max_pool(C2, [1,2,1,1] , [1,2,1,1], padding='VALID')
        C2 = tf.expand_dims(tf.squeeze(C2,[2]), -1)

        F3 = tf.Variable(tf.random_normal([self.kernel, self.n_filters[1] ,1, self.n_filters[2]] ,-0.05,
0.05),name = 'F3')
        FB3 = tf.Variable(tf.constant(0.1, shape=[self.n_filters[2]]),name = 'FB3')
        C3 = tf.nn.relu(tf.add(tf.nn.conv2d(C2, F3, [1,1,1,1], padding='VALID'), FB3))
        C3 = tf.nn.max_pool(C3, [1,2,1,1] , [1,2,1,1], padding='VALID')
        C3 = tf.expand_dims(tf.squeeze(C3,[2]), -1)

        F4 = tf.Variable(tf.random_normal([self.kernel, self.n_filters[2] ,1, self.n_filters[3]] ,-0.05,
0.05),name = 'F4')
        FB4 = tf.Variable(tf.constant(0.1, shape=[self.n_filters[3]]),name = 'FB4')
        C4 = tf.nn.relu(tf.add(tf.nn.conv2d(C3, F4, [1,1,1,1], padding='VALID'), FB4))
        C4 = tf.nn.max_pool(C4, [1,C4.get_shape()[1],1,1] , [1,1,1,1], padding='VALID')
        C4 = tf.squeeze(C4,[1,2])
        C4 = tf.tile(C4,[self.beam_width,1])
        C4 = tf.expand_dims(C4, 0)
        #state1 = tf.zeros([self.batch_size, self.lstm1.state_size])
        #state2 = tf.zeros([self.batch_size, self.lstm2.state_size])
        #padding = tf.zeros([self.batch_size, self.dim_hidden])

        generated_words = []

        probs = []
        embeds = []

        with tf.variable_scope("bilstm1"):
            bi_outputs, (encoder_state_fw, encoder_state_bw) =
tf.nn.bidirectional_dynamic_rnn(self.lstm1,
                                                    self.lstm2,
                                                    image_emb,
                                                    dtype = tf.float32,
                                                    time_major=False)

        encoder_outputs = tf.concat(bi_outputs, -1)
        #encoder_state = tf.concat(encoder_state, -1)
        #encoder_state_c = (encoder_state_fw.c + encoder_state_bw.c)/2
        encoder_state_c = tf.tile(lstm_c,[self.beam_width, 1])
```

```
        #encoder_state_h = (encoder_state_fw.h + encoder_state_bw.h)/2
        encoder_state_h = tf.tile(lstm_h,[self.beam_width, 1])
        encoder_state = tf.nn.rnn_cell.LSTMStateTuple(encoder_state_c, encoder_state_h)

        with tf.variable_scope("bilstm2"):
            bi_outputs_video, (encoder_state_fw_video, encoder_state_bw_video) = tf.nn.bidirectional_dynamic_rnn(self.lstm4,
                                                                                                                 self.lstm5,
                                                                                                                 video_emb,
                                                                                                                 dtype = tf.float32,
                                                                                                                 time_major=False)



        encoder_outputs_video = tf.concat(bi_outputs_video, -1)



        def embed_and_input_proj(inputs):
            a = tf.nn.embedding_lookup(self.Wemb, inputs)
            return tf.concat([a, C4], 2)

        attention_states = tf.transpose(encoder_outputs, [0, 1, 2])
        attention_states = tf.contrib.seq2seq.tile_batch(attention_states, multiplier=beam_width)
        attention_states_video = tf.transpose(encoder_outputs_video, [0,1,2])
        attention_states_video = tf.contrib.seq2seq.tile_batch(attention_states_video, multiplier=beam_width)

        video_3d_1 = tf.contrib.seq2seq.tile_batch(video_3d, multiplier=beam_width)

        start = tf.fill([1], tf.constant(1,dtype = tf.int32))
        end = tf.constant(20000,dtype = tf.int32)

        #helper = tf.contrib.seq2seq.GreedyEmbeddingHelper(start_tokens=start,
        #                                                  end_token=end,
        #                                                  embedding=embed_and_input_proj)
        with tf.variable_scope("ATT"):
            attention_mechanism = tf.contrib.seq2seq.BahdanauAttention(dim_hidden,attention_states)
            attention_mechanism_video = tf.contrib.seq2seq.BahdanauAttention(dim_hidden,video_3d_1)

        #state3 = lstm3.zero_state(self.batch_size)
```

```
                projection_layer = layers_core.Dense(self.n_words, use_bias=False)

                decoder = tf.contrib.seq2seq.AttentionWrapper(self.lstm3,
[attention_mechanism,attention_mechanism_video],
attention_layer_size=[dim_hidden,dim_hidden])
                #decoder = tf.contrib.seq2seq.AttentionWrapper(decoder,
attention_mechanism_video, attention_layer_size=dim_hidden)

            #state3 = decoder.zero_state(batch_size=1, dtype=tf.float32)
                state3 = decoder.zero_state(batch_size=beam_width, dtype=tf.float32)
                state3 = state3.clone(cell_state=encoder_state)

            #decoder = tf.contrib.rnn.DropoutWrapper(decoder, input_keep_prob = 0.5,
output_keep_prob = 0.5)
            #decoder_cell = tf.contrib.seq2seq.BasicDecoder(decoder, helper,
state3 ,output_layer=projection_layer)

                decoder_cell = tf.contrib.seq2seq.BeamSearchDecoder(decoder,
                                                    embedding=embed_and_input_proj,
                                                    start_tokens=start,
                                                    end_token=end,
                                                    initial_state=state3,
                                                    beam_width = beam_width,
                                                    output_layer=projection_layer)

            #decoder = tf.contrib.seq2seq.AttentionWrapper(self.lstm3, attention_mechanism,
attention_layer_size=dim_hidden)
                outputs, _, _ = tf.contrib.seq2seq.dynamic_decode(decoder_cell,
                        output_time_major=False,
                        #impute_finished=True,
                        maximum_iterations=n_caption_lstm_step)

            generated_words = tf.squeeze(outputs.predicted_ids[0])

            #generated_words = tf.squeeze(tf.multinomial(outputs.rnn_output, -1))
            return video, video_mask, generated_words, video_3d, audio ,cate #, probs,
embeds

#======================================================================
=======
# Global Parameters
#======================================================================
=======
video_path = '/data1/jintao/YouTubeClips'
```

```
video_train_feat_path = '/share/msrvtt_merge/temporal_train'
video_test_feat_path = '/share/msrvtt_merge/temporal_test'
video_val_feat_path = '/share/msrvtt_merge/temporal_val'

video_train_3d_path = '/share/msrvtt_merge/i3d_train'
video_test_3d_path = '/share/msrvtt_merge/i3d_test'
video_val_3d_path = '/share/msrvtt_merge/i3d_val'

video_train_mfcc_path = '/share/msrvtt_merge/mfcc_train'
video_test_mfcc_path = '/share/msrvtt_merge/mfcc_test'
video_val_mfcc_path = '/share/msrvtt_merge/mfcc_val'

video_train_cate_path = '/share/msrvtt_merge/category_train'
video_test_cate_path = '/share/msrvtt_merge/category_test'
video_val_cate_path = '/share/msrvtt_merge/category_val'

video_train_data_path = '/data1/jintao/msrvtt_file/coco_eval/msrvtt_data.csv'
video_test_data_path = '/data1/jintao/msrvtt_file/coco_eval/msrvtt_data.csv'
video_val_data_path = '/data1/jintao/msrvtt_file/coco_eval/msrvtt_data.csv'

model_path = '/data1/jintao/msrvtt_file/train_model/i3d_cate_similarity'

#=============================================================================
==========
# Train Parameters
#=============================================================================
==========
dim_image = 2560
dim_hidden= 512

dim_video = 2048
n_3d_lstm_step = 50

n_video_lstm_step = 80
n_caption_lstm_step = 35
n_frame_step = 80

n_epochs = 101
batch_size = 50
initial_learning_rate = 0.0001
l2_rate = 0.005
beam_width = 5
drop_prob = 0.5
```

```
kernel = 5
n_filters = [192,256,512,512]


def get_video_train_data(video_data_path, video_feat_path, video_3d_path, video_mfcc_path, video_cate_path):
    video_data = pd.read_csv(video_data_path, sep=',')
    #video_data = video_data[video_data['Language'] == 'English']
    #video_data['video_path'] = video_data.apply(lambda row: row['VideoID']+'_'+str(int(row['Start']))+'_'+str(int(row['End']))+'.avi.npy', axis=1)
    video_data['video_path'] = video_data.apply(lambda row: row['video_id'] + '.mp4.npy', axis=1)
    #video_data['ori'] = video_data.apply(lambda row: row['VideoID']+'_'+str(int(row['Start']))+'_'+str(int(row['End'])), axis=1)
    video_data['ori'] = video_data.apply(lambda row: row['video_id'], axis=1)

    video_data['video_3d'] = video_data['video_path'].map(lambda x: os.path.join(video_3d_path, x))
    video_data['video_mfcc'] = video_data['video_path'].map(lambda x: os.path.join(video_mfcc_path, x))
    video_data['video_cate'] = video_data['video_path'].map(lambda x: os.path.join(video_cate_path, x))

    video_data['video_path'] = video_data['video_path'].map(lambda x: os.path.join(video_feat_path, x))
    #video_data['video_mfcc'] = video_data['video_path'].map(lambda x: os.path.join(video_mfcc_path, x))

    video_data = video_data[video_data['video_path'].map(lambda x: os.path.exists( x ))]
    video_data = video_data[video_data['caption'].map(lambda x: isinstance(x, str))]

    unique_filenames = sorted(video_data['video_path'].unique())
    train_data = video_data[video_data['video_path'].map(lambda x: x in unique_filenames)]
    return train_data

def get_video_test_data(video_data_path, video_feat_path, video_3d_path, video_mfcc_path, video_cate_path):
    video_data = pd.read_csv(video_data_path, sep=',')
    #video_data = video_data[video_data['Language'] == 'English']
    #video_data['video_path'] = video_data.apply(lambda row: row['VideoID']+'_'+str(int(row['Start']))+'_'+str(int(row['End']))+'.avi.npy', axis=1)
    video_data['video_path'] = video_data.apply(lambda row: row['video_id'] + '.mp4.npy', axis=1)
    #video_data['ori'] = video_data.apply(lambda row:
```

```
row['VideoID']+'_'+str(int(row['Start']))+'_'+str(int(row['End'])), axis=1)
    video_data['ori'] = video_data.apply(lambda row: row['video_id'], axis=1)

    video_data['video_3d'] = video_data['video_path'].map(lambda x: os.path.join(video_3d_path, x))
    video_data['video_mfcc'] = video_data['video_path'].map(lambda x: os.path.join(video_mfcc_path, x))
    video_data['video_cate'] = video_data['video_path'].map(lambda x: os.path.join(video_cate_path, x))

    video_data['video_path'] = video_data['video_path'].map(lambda x: os.path.join(video_feat_path, x))
    #video_data['video_mfcc'] = video_data['video_path'].map(lambda x: os.path.join(video_mfcc_path, x))

    video_data = video_data[video_data['video_path'].map(lambda x: os.path.exists( x ))]
    video_data = video_data[video_data['caption'].map(lambda x: isinstance(x, str))]

    unique_filenames = sorted(video_data['video_path'].unique())
    test_data = video_data[video_data['video_path'].map(lambda x: x in unique_filenames)]
    return test_data

def get_video_val_data(video_data_path, video_feat_path, video_3d_path, video_mfcc_path, video_cate_path):
    video_data = pd.read_csv(video_data_path, sep=',')
    #video_data = video_data[video_data['Language'] == 'English']
    #video_data['video_path'] = video_data.apply(lambda row: row['VideoID']+'_'+str(int(row['Start']))+'_'+str(int(row['End']))+'.avi.npy', axis=1)
    video_data['video_path'] = video_data.apply(lambda row: row['video_id'] + '.mp4.npy', axis=1)
    #video_data['ori'] = video_data.apply(lambda row: row['VideoID']+'_'+str(int(row['Start']))+'_'+str(int(row['End'])), axis=1)
    video_data['ori'] = video_data.apply(lambda row: row['video_id'], axis=1)

    video_data['video_3d'] = video_data['video_path'].map(lambda x: os.path.join(video_3d_path, x))
    video_data['video_mfcc'] = video_data['video_path'].map(lambda x: os.path.join(video_mfcc_path, x))
    video_data['video_cate'] = video_data['video_path'].map(lambda x: os.path.join(video_cate_path, x))

    video_data['video_path'] = video_data['video_path'].map(lambda x: os.path.join(video_feat_path, x))
    #video_data['video_mfcc'] = video_data['video_path'].map(lambda x:
```

```
os.path.join(video_mfcc_path, x))

    video_data = video_data[video_data['video_path'].map(lambda x: os.path.exists( x ))]
    video_data = video_data[video_data['caption'].map(lambda x: isinstance(x, str))]

    unique_filenames = sorted(video_data['video_path'].unique())
    val_data = video_data[video_data['video_path'].map(lambda x: x in unique_filenames)]
    return val_data


def preProBuildWordVocab(sentence_iterator, word_count_threshold=5):
    # borrowed this function from NeuralTalk
    print 'preprocessing word counts and creating vocab based on word count threshold %d' %
(word_count_threshold)
    word_counts = {}
    word_counts2 = {}
    nsents = 0
    for sent in sentence_iterator:
        nsents += 1
        for w in sent.lower().split(' '):
            word_counts[w] = word_counts.get(w, 0) + 1
    vocab = [w for w in word_counts if word_counts[w] >= word_count_threshold]
    print 'filtered words from %d to %d' % (len(word_counts), len(vocab))

    for xxx in vocab:
        word_counts2[xxx] = word_counts[xxx]

    ixtoword = {}
    ixtoword[0] = '<pad>'
    ixtoword[1] = '<bos>'
    ixtoword[2] = '<eos>'
    ixtoword[3] = '<unk>'

    wordtoix = {}
    wordtoix['<pad>'] = 0
    wordtoix['<bos>'] = 1
    wordtoix['<eos>'] = 2
    wordtoix['<unk>'] = 3

    for idx, w in enumerate(vocab):
        wordtoix[w] = idx+4
        ixtoword[idx+4] = w

    word_counts['<pad>'] = nsents
```

```
    word_counts['<bos>'] = nsents
    word_counts['<eos>'] = nsents
    word_counts['<unk>'] = nsents

    word_counts2['<pad>'] = nsents
    word_counts2['<bos>'] = nsents
    word_counts2['<eos>'] = nsents
    word_counts2['<unk>'] = nsents


    bias_init_vector = np.array([1.0 * word_counts[ ixtoword[i] ] for i in ixtoword])
    bias_init_vector /= np.sum(bias_init_vector) # normalize to frequencies
    bias_init_vector = np.log(bias_init_vector)
    bias_init_vector -= np.max(bias_init_vector) # shift to nice numeric range

    return wordtoix, ixtoword, bias_init_vector, word_counts2



def get_validation_bleu(sess, current_val_data, ixtoword, wordtoix, video_tf, video_mask_tf, generated_words_tf,train_data,video_3d_tf, audio_tf, cate_tf, hidden_out_tf):

    loss_vall = []
    ixtoword2 = pd.Series(np.load('./data/ixtoword.npy').tolist())
    count = 0
    val_data = current_val_data
    reward = np.zeros((batch_size,n_caption_lstm_step))
    diss = {}
    a = {}
    for start, end in zip(
                range(0, len(val_data), batch_size),
                range(batch_size, len(val_data), batch_size)):

            start_time = time.time()

            current_batch = val_data[start:end]
            current_videos = current_batch['video_path'].values
            current_concate = current_batch['ori'].values
            current_i3d = current_batch['video_3d'].values
            current_mfcc = current_batch['video_mfcc'].values
            current_cate = current_batch['video_cate'].values

            #current_concate_feats = []
            #for i, val in enumerate(current_concate):
```

```
                #       current_concate[i] = mapID[val]
                #       current_concate_feat = data_feat[mapID[val]-1]
                #       current_concate_feats.append(current_concate_feat)

                current_feats = np.zeros((batch_size, n_video_lstm_step, dim_image))
                current_feats_vals = map(lambda vid: np.load(vid), current_videos)
                current_i3d_feats = np.zeros((batch_size, n_3d_lstm_step, dim_video))
                current_feats_i3d = map(lambda vid: np.load(vid), current_i3d)
                current_mfcc_feats = np.zeros((batch_size, 80, 9984))
                current_feats_mfcc = map(lambda vid: np.load(vid), current_mfcc)

                current_cate_feats = np.zeros((batch_size, 80, 300))
                current_feats_cate = map(lambda vid: np.load(vid), current_cate)


                current_video_masks = np.zeros((batch_size, n_video_lstm_step))
#                 current_video_masks = np.zeros((batch_size, n_video_lstm_step))

                for ind,feat in enumerate(current_feats_vals):
                        current_feats[ind][:len(current_feats_vals[ind])] = feat
                        current_video_masks[ind][:len(current_feats_vals[ind])] = 1

                for ind,feat in enumerate(current_feats_i3d):
                        current_i3d_feats[ind][:len(current_feats_i3d[ind])] = feat

                for ind,feat in enumerate(current_feats_mfcc):
                        current_mfcc_feats[ind][:len(current_feats_mfcc[ind])] = feat

                for ind,feat in enumerate(current_feats_cate):
                        current_cate_feats[ind][:len(current_feats_cate[ind])] = feat

                generated_word_index,                   hidden_out                      =
sess.run([generated_words_tf,hidden_out_tf],            feed_dict={video_tf:current_feats,
video_mask_tf:current_video_masks,                          video_3d_tf:current_i3d_feats,
audio_tf:current_mfcc_feats, cate_tf:current_cate_feats})
                generated_words = []
                #from IPython import embed; embed()

                #print generated_word_index[0]
                #for i in range(n_caption_lstm_step):
                #       aa = list(generated_word_index[i])
                #       generated_words.append(aa)
                #       generated_words2 = map(list,zip(*generated_words))
```

```
for i in range(batch_size):
    currentt_videos = train_data[train_data['video_path'] == current_videos[i]]


    currentt_captions = currentt_videos['caption'].values
    #from IPython import embed; embed()
    currentt_captions = map(lambda x: x.replace('.', ''), currentt_captions)
    currentt_captions = map(lambda x: x.replace(',', ''), currentt_captions)
    currentt_captions = map(lambda x: x.replace('"', ''), currentt_captions)
    currentt_captions = map(lambda x: x.replace('\n', ''), currentt_captions)
    currentt_captions = map(lambda x: x.replace('?', ''), currentt_captions)
    currentt_captions = map(lambda x: x.replace('!', ''), currentt_captions)
    currentt_captions = map(lambda x: x.replace('\\', ''), currentt_captions)
    currentt_captions = map(lambda x: x.replace('/', ''), currentt_captions)
    currentt_captions = map(lambda x: x.replace('    ', ' '), currentt_captions)

    #currentt_captions = map(lambda x: x.lower(), currentt_captions)
    for idx, each_cap in enumerate(currentt_captions):
        word = each_cap.lower()
        if word[0] == '':
            word = word[1:]


        currentt_captions[idx] = word                    #dui zhengge danci caozuo



    #diss[1] = list(currentt_videos['Description'].values)
    #diss[count] = list(currentt_videos['Description'].values)
    diss[count] = list(currentt_captions)

    gen = ixtoword2[generated_word_index[i]]
    punctuation = np.argmax(np.array(gen) == '<eos>') + 1
    gen = gen[:punctuation]

    generat_sentence = ' '.join(gen)
    generat_sentence = generat_sentence.replace('<bos> ', '')
    generat_sentence = generat_sentence.replace(' <eos>', '')
    #a[1] = [generat_sentence]
    a[count] = [generat_sentence]
    #if i==batch_size/2:
    #print generat_sentence

    #print diss[0]
```

```
                    #from IPython import embed; embed()
                    #ward, _ = Rouge().compute_score(diss, a)
                    #red.append(ward)
                    count = count + 1

                #print str(time.time()-start_time)
                    #reward[i][j] =ward


        #print a
        print count
        ward, _ = Cider().compute_score(diss, a)
        return ward



def train():
        #with tf.variable_scope(tf.get_variable_scope(),reuse=False):
        train_data = get_video_train_data(video_train_data_path, video_train_feat_path,
video_train_3d_path, video_train_mfcc_path, video_train_cate_path)
        train_captions = train_data['caption'].values
        test_data = get_video_test_data(video_test_data_path, video_test_feat_path,
video_test_3d_path, video_test_mfcc_path, video_test_cate_path)
        test_captions = test_data['caption'].values

        val_data = get_video_val_data(video_val_data_path, video_val_feat_path,
video_val_3d_path, video_val_mfcc_path, video_val_cate_path)
        val_captions = val_data['caption'].values

        current_epoch = tf.Variable(0)
        captions_list = list(train_captions) + list(val_captions)
        captions = np.asarray(captions_list, dtype=np.object)

        captions = map(lambda x: x.replace('.', ''), captions)
        captions = map(lambda x: x.replace(',', ''), captions)
        captions = map(lambda x: x.replace('"', ''), captions)
        captions = map(lambda x: x.replace('\n', ''), captions)
        captions = map(lambda x: x.replace('?', ''), captions)
        captions = map(lambda x: x.replace('!', ''), captions)
        captions = map(lambda x: x.replace('\\', ''), captions)
        captions = map(lambda x: x.replace('/', ''), captions)
        captions = map(lambda x: x.replace('    ', ' '), captions)
```

```
    wordtoix, ixtoword, bias_init_vector, word_counts = preProBuildWordVocab(captions, word_count_threshold=3)

    w_file = '/data1/jintao/s2vt/GoogleNews-vectors-negative300.bin'

    w2v = load_bin_vec(w_file, word_counts)
    add_unknown_words(w2v, word_counts)

    w = np.zeros((len(w2v.keys()), 300), dtype = 'float32')
    for word in w2v:
        w[wordtoix[word]] = w2v[word]

    print w.shape

    np.save("./data/wordtoix", wordtoix)
    np.save('./data/ixtoword', ixtoword)
    np.save("./data/bias_init_vector", bias_init_vector)

    with tf.variable_scope(tf.get_variable_scope(),reuse=False) as scope:
        model = Video_Caption_Generator(
            dim_image=dim_image,
            n_words=len(wordtoix),
            dim_hidden=dim_hidden,
            batch_size=batch_size,
            n_lstm_steps=n_frame_step,
            n_video_lstm_step=n_video_lstm_step,
            n_caption_lstm_step=n_caption_lstm_step,
            beam_width = beam_width,
            bias_init_vector=bias_init_vector,
            drop_prob = drop_prob,
            kernel = kernel,
            n_filters = n_filters,
            dim_video = dim_video,
            n_3d_lstm_step = n_3d_lstm_step
            )

        tf_loss, tf_video, tf_video_mask, tf_caption, tf_caption_mask, tf_reward, tf_flag, tf_prob, tf_video_3d, tf_audio, tf_cate, tf_embed_placeholder, tf_embed_init, tf_senten = model.build_model()

        scope.reuse_variables()
        video_tf, video_mask_tf, generated_words_tf, video_3d_tf, audio_tf, cate_tf, hidden_out_tf = model.build_generator()
```

```
        sess = tf.InteractiveSession()

        # my tensorflow version is 0.12.1, I write the saver with version 1.0
        saver = tf.train.Saver(max_to_keep=100, write_version=1)

        learning_rate = tf.train.exponential_decay(initial_learning_rate,
                                                   global_step=current_epoch,
                                                   decay_steps=1000,decay_rate=0.5,staircase
= True)

        train_op = tf.train.AdamOptimizer(learning_rate).minimize(loss=tf_loss)
        tf.global_variables_initializer().run()

        sess.run(tf_embed_init, feed_dict = {tf_embed_placeholder: w} )

        #new_saver = tf.train.Saver()
        #new_saver = tf.train.import_meta_graph('./rgb_models/model-1000.meta')
        #new_saver.restore(sess, tf.train.latest_checkpoint('./models/'))

#saver.restore(sess,'/data1/jintao/msrvtt_file/train_model/i3d_cate_vse2/i3d_cate_vse2-78')

        loss_fd = open('./result_txt/i3d_cate_similarity.txt', 'w')
        #mapVT = open('map.txt','r')
        #mapID = {}
        #ID = 1
        #for lines in mapVT:
        #       tmp = lines.strip().split(' ')
        #       mapID[tmp[0]] = ID
        #       ID = ID + 1
        #data = scipy.io.loadmat('tag_feats.mat')
        #data_feat = data['feats']

        loss_to_draw = []
        val_loss = []
        reward = np.zeros((batch_size,n_caption_lstm_step))
        for epoch in range(0, n_epochs):
                current_epoch = epoch
                loss_to_draw_epoch = []

                index = list(train_data.index)
                np.random.shuffle(index)
                train_data = train_data.ix[index]
                #from IPython import embed; embed()
```

```
        current_train_data = train_data.groupby('video_path').apply(lambda x: x.iloc[np.random.choice(len(x)) ] )
        current_train_data = current_train_data.reset_index(drop=True)
        for start, end in zip(
                range(0, len(current_train_data), batch_size),
                range(batch_size, len(current_train_data), batch_size)):

            start_time = time.time()

            current_batch = current_train_data[start:end]

            #print str(time.time() - start_time)
            #from IPython import embed; embed()
            current_videos = current_batch['video_path'].values
            current_concate = current_batch['ori'].values
            current_i3d = current_batch['video_3d'].values
            current_mfcc = current_batch['video_mfcc'].values
            current_cate = current_batch['video_cate'].values
            #current_concate_feats = []
            #for i, val in enumerate(current_concate):
            #    current_concate[i] = mapID[val]
            #    current_concate_feat = data_feat[mapID[val]-1]
            #    current_concate_feats.append(current_concate_feat)



            current_feats = np.zeros((batch_size, n_video_lstm_step, dim_image))
            current_feats_vals = map(lambda vid: np.load(vid), current_videos) #buzu 80 de buling bing biaoji wei 0
            current_i3d_feats = np.zeros((batch_size, n_3d_lstm_step, dim_video))
            current_feats_i3d = map(lambda vid: np.load(vid), current_i3d)
            current_mfcc_feats = np.zeros((batch_size, 80, 9984))
            current_feats_mfcc = map(lambda vid: np.load(vid), current_mfcc)

            current_cate_feats = np.zeros((batch_size, 80, 300))
            current_feats_cate = map(lambda vid: np.load(vid), current_cate)

            #from IPython import embed; embed()



            current_video_masks = np.zeros((batch_size, n_video_lstm_step))

            for ind,feat in enumerate(current_feats_vals):
```

```
                current_feats[ind][:len(current_feats_vals[ind])] = feat
                current_video_masks[ind][:len(current_feats_vals[ind])] = 1

            for ind,feat in enumerate(current_feats_i3d):
                current_i3d_feats[ind][:len(current_feats_i3d[ind])] = feat

            for ind,feat in enumerate(current_feats_mfcc):
                current_mfcc_feats[ind][:len(current_feats_mfcc[ind])] = feat

            for ind,feat in enumerate(current_feats_cate):
                current_cate_feats[ind][:len(current_feats_cate[ind])] = feat

            current_captions = current_batch['caption'].values
            current_captions = map(lambda x: '<bos> ' + x, current_captions)
            current_captions = map(lambda x: x.replace('.', ''), current_captions)
            current_captions = map(lambda x: x.replace(',', ''), current_captions)
            current_captions = map(lambda x: x.replace('"', ''), current_captions)
            current_captions = map(lambda x: x.replace('\n', ''), current_captions)
            current_captions = map(lambda x: x.replace('?', ''), current_captions)
            current_captions = map(lambda x: x.replace('!', ''), current_captions)
            current_captions = map(lambda x: x.replace('\\', ''), current_captions)
            current_captions = map(lambda x: x.replace('/', ''), current_captions)
            current_captions = map(lambda x: x.replace('    ', ' '), current_captions)

            #print str(time.time() - start_time)
            #from IPython import embed; embed()

            for idx, each_cap in enumerate(current_captions):
                word = each_cap.lower().split(' ')
                if word[0] == '':
                    word = word[1:]

                if len(word) < n_caption_lstm_step:
                    current_captions[idx] = current_captions[idx] + ' <eos>'
#dui zhengge danci caozuo
                else:
                    new_word = ''
                    for i in range(n_caption_lstm_step-1):
                        new_word = new_word + word[i] + ' '
                    current_captions[idx] = new_word + '<eos>'

            current_caption_ind = []
            for cap in current_captions:
                current_word_ind = []
```

```
                for word in cap.lower().split(' '):
                    if word in wordtoix:
                        current_word_ind.append(wordtoix[word])
                    else:
                        current_word_ind.append(wordtoix['<unk>'])
                current_caption_ind.append(current_word_ind)                    #ind shi jiang
zhengshu fangru!

            current_caption_matrix = sequence.pad_sequences(current_caption_ind,
padding='post', maxlen=n_caption_lstm_step)
            current_caption_matrix = np.hstack( [current_caption_matrix,
np.zeros( [len(current_caption_matrix), 1] ) ] ).astype(int)
            current_caption_masks = np.zeros( (current_caption_matrix.shape[0],
current_caption_matrix.shape[1]) )
            nonzeros = np.array( map(lambda x: (x != 0).sum() + 1, current_caption_matrix ) )

            for ind, row in enumerate(current_caption_masks):
                row[:nonzeros[ind]] = 1

            #print str(time.time() - start_time)
            #from IPython import embed; embed()

            generated_word_index,                          hidden_out                    =
sess.run([generated_words_tf,hidden_out_tf],                feed_dict={video_tf:current_feats,
video_mask_tf:current_video_masks,                          video_3d_tf:current_i3d_feats,
audio_tf:current_mfcc_feats, cate_tf:current_cate_feats})
            print generated_word_index[0]


            _, loss_val, senten = sess.run(
                    [train_op, tf_loss, tf_senten],
                    feed_dict={
                        tf_video: current_feats,
                        tf_video_mask : current_video_masks,
                        tf_caption: current_caption_matrix,
                        tf_caption_mask: current_caption_masks,
                        tf_reward: reward,
                        tf_flag: 0,
                        tf_prob:0.5,
                        #tf_concate:current_concate_feats,
                        tf_video_3d:current_i3d_feats,
                        tf_audio:current_mfcc_feats,
                        tf_cate:current_cate_feats
                        })
```

```
            loss_to_draw_epoch.append(loss_val)

            #print str(time.time() - start_time)
            #from IPython import embed; embed()

            print 'idx: ', start, " Epoch: ", epoch, " loss: ", loss_val, ' Elapsed time: ', str((time.time() - start_time))
            #loss_fd.write('epoch ' + str(epoch) + ' loss ' + str(loss_val) + '\n')
            print senten[0]

            #generated_word_index, hidden_out = sess.run([generated_words_tf,hidden_out_tf], feed_dict={video_tf:current_feats, video_mask_tf:current_video_masks, video_3d_tf:current_i3d_feats, audio_tf:current_mfcc_feats, cate_tf:current_cate_feats})
            #print senten[0]



        # draw loss curve every epoch
        #val_loss_epoch = get_validation_loss(sess,
        #                                     val_data.groupby('video_path').apply(lambda x: x.iloc[np.random.choice(len(x))]).reset_index(drop=True),
        #                    ixtoword,
        #                          wordtoix, tf_loss, tf_video, tf_caption, tf_caption_mask, tf_video_mask, tf_reward, tf_flag, tf_prob)
        #val_bleu_epoch = get_validation_bleu(sess,
        #                                     val_data.groupby('video_path').apply(lambda x: x.iloc[np.random.choice(len(x))]).reset_index(drop=True),
        #                    ixtoword,wordtoix,
        #                            video_tf, video_mask_tf, generated_words_tf,val_data, video_3d_tf, audio_tf, cate_tf, hidden_out_tf)
        #loss_fd.write('epoch ' + str(epoch) + ' loss ' + str(val_bleu_epoch) + '\n')

        #print 'Epoch:    ', epoch, 'val_loss:    ',val_loss_epoch
        #print 'Epoch:    ', epoch, 'val_bleu:    ',val_bleu_epoch

        #loss_to_draw.append(np.mean(loss_to_draw_epoch))
        #val_loss.append(val_loss_epoch)
        #val_loss.append(val_loss_epoch)

        #plt_save_dir = "./loss_rebilstm"
        #plt_save_img_name = str(epoch) + '.png'

        #xxx = range(len(loss_to_draw))
```

```
            #plt.plot(xxx, loss_to_draw, color='g')
            #plt.plot(xxx, val_loss, color='r')

            #plt.grid(True)
            #if np.mod(epoch,100) == 0:
            #       plt.savefig(os.path.join(plt_save_dir, plt_save_img_name))

            #if np.mod(epoch, 10) == 0:
                  #val_loss = get_validation_loss(sess,
                                    #val_data.groupby('video_path').apply(lambda x:
x.iloc[np.random.choice(len(x))]).reset_index(drop=True),
                                    #ixtoword,
                                    #wordtoix,    tf_loss,    tf_video,    tf_caption,    tf_caption_mask,
tf_video_mask, tf_reward, tf_flag)
                  #print 'Epoch: ', epoch, 'val_loss:    ',val_loss

            #if val_bleu_epoch>=0.43 or np.mod(epoch, 10) == 0:
            #       test_bleu_epoch = get_validation_bleu(sess,
            #                                             test_data.groupby('video_path').apply(lambda   x:
x.iloc[np.random.choice(len(x))]).reset_index(drop=True),
            #                           ixtoword,wordtoix,
            #                                       video_tf,  video_mask_tf,  generated_words_tf,test_data,
video_3d_tf, audio_tf, cate_tf, hidden_out_tf)
            #       print 'Epoch:    ', epoch, 'test_bleu:    ',test_bleu_epoch
            #      loss_fd.write('epoch ' + str(epoch) + ' loss ' + str(test_bleu_epoch) + '\n')

            #       if test_bleu_epoch >= 0.45:
            #             print "Epoch ", epoch, " is done. Saving the model ..."
            #                         saver.save(sess,   os.path.join(model_path,   'i3d_cate_similarity'),
global_step=epoch)


      #loss_fd.close()



def test(model_path='/data1/jintao/msrvtt_file/train_model/i3d_cate_vse6/i3d_cate_vse6-88'):
      test_data       =       get_video_test_data(video_test_data_path,       video_test_feat_path,
video_test_3d_path, video_test_mfcc_path, video_test_cate_path)
      test_videos = test_data['video_path'].unique()
      concate_videos = test_data['ori'].unique()

      ixtoword = pd.Series(np.load('./data/ixtoword.npy').tolist())
```

```
    bias_init_vector = np.load('./data/bias_init_vector.npy')

    #mapVT = open('map.txt','r')
    #mapID = {}
    #ID = 1
    #for lines in mapVT:
    #       tmp = lines.strip().split(' ')
    #       mapID[tmp[0]] = ID
    #       ID = ID + 1
    #data = scipy.io.loadmat('tag_feats.mat')
    #data_feat = data['feats']

    model = Video_Caption_Generator(
                dim_image=dim_image,
                n_words=len(ixtoword),
                dim_hidden=dim_hidden,
                batch_size=batch_size,
                n_lstm_steps=n_frame_step,
                n_video_lstm_step=n_video_lstm_step,
                n_caption_lstm_step=n_caption_lstm_step,
                beam_width = beam_width,
                bias_init_vector=bias_init_vector,
                drop_prob = drop_prob,
                kernel = kernel,
                n_filters = n_filters,
                dim_video = dim_video,
                n_3d_lstm_step = n_3d_lstm_step
                        )

    video_tf, video_mask_tf, caption_tf, video_3d_tf, audio_tf, cate_tf =
model.build_generator_single()

    sess = tf.InteractiveSession()

    #tf.global_variables_initializer().run()
    saver = tf.train.Saver()
    saver.restore(sess, model_path)

    test_output_txt_fd = open('bi_results.txt', 'w')
    for idx, video_feat_path in enumerate(concate_videos):
        print idx, video_feat_path

        #concate_data = data_feat[mapID[video_feat_path]-1]
        #concate_data = np.expand_dims(concate_data, 0)
```

```
            #concate_data = np.tile(concate_data, (beam_width, 1))

            #concate_data = np.expand_dims(concate_data, 0)
            #print concate_data.shape
            video_feat = np.load(video_test_feat_path + '/'+video_feat_path+'.mp4.npy')[None,...]

            video_i3d_feat = np.load(video_test_3d_path + '/'+video_feat_path+'.mp4.npy')
            video_i3d_feat = np.expand_dims(video_i3d_feat, 0)
            shape_templete2 = np.zeros(shape=(1, n_3d_lstm_step, dim_video), dtype=float )

shape_templete2[:video_i3d_feat.shape[0],  :video_i3d_feat.shape[1],  :video_i3d_feat.shape[2]] = video_i3d_feat
            video_i3d_feat = shape_templete2

            video_mfcc_feat = np.load(video_test_mfcc_path + '/'+video_feat_path+'.mp4.npy')
            video_mfcc_feat = np.expand_dims(video_mfcc_feat, 0)
            shape_templete3 = np.zeros(shape=(1, 80, 9984), dtype=float )

shape_templete3[:video_mfcc_feat.shape[0],  :video_mfcc_feat.shape[1],  :video_mfcc_feat.shape[2]] = video_mfcc_feat
            video_mfcc_feat = shape_templete3

            video_cate_feat = np.load(video_test_cate_path + '/'+video_feat_path+'.mp4.npy')
            video_cate_feat = np.expand_dims(video_cate_feat, 0)
            shape_templete4 = np.zeros(shape=(1, 80, 300), dtype=float )

shape_templete4[:video_cate_feat.shape[0],  :video_cate_feat.shape[1],  :video_cate_feat.shape[2]] = video_cate_feat
            video_cate_feat = shape_templete4

            #video_feat = np.load(video_feat_path)
            #video_mask = np.ones((video_feat.shape[0], video_feat.shape[1]))
            if video_feat.shape[1] == n_frame_step:
                video_mask = np.ones((video_feat.shape[0], video_feat.shape[1]))
            else:
                #continue
                shape_templete = np.zeros(shape=(1, n_frame_step, dim_image), dtype=float )
                shape_templete[:video_feat.shape[0],  :video_feat.shape[1],  :video_feat.shape[2]] = video_feat
                video_feat = shape_templete



                video_mask = np.zeros((video_feat.shape[0], n_frame_step))
```

```
            video_dd      = np.ones((video_feat.shape[0],video_feat.shape[1]))
            video_mask[:video_feat.shape[0], :video_feat.shape[1]] = video_dd
            #from IPython import embed; embed()
        generated_word_index = sess.run(caption_tf, feed_dict={video_tf:video_feat, video_mask_tf:video_mask,video_3d_tf:video_i3d_feat, audio_tf:video_mfcc_feat, cate_tf:video_cate_feat})
        #from IPython import embed; embed()

        generated_words = ixtoword[generated_word_index[:,0]]
        #generated_words = ixtoword[generated_word_index]
        punctuation = np.argmax(np.array(generated_words) == '<eos>') + 1
        generated_words = generated_words[:punctuation]

        generated_sentence = ' '.join(generated_words)
        generated_sentence = generated_sentence.replace('<bos> ', '')
        generated_sentence = generated_sentence.replace(' <eos>', '')
        print generated_sentence,'\n'
        test_output_txt_fd.write(video_test_feat_path + '/'+video_feat_path+'.mp4.npy' + '\n')
        test_output_txt_fd.write(generated_sentence + '\n\n')
```