```
# Author: Shashi Narayan
# Date: September 2016
# Project: Document Summarization
# H2020 Summa Project
#####################################

"""
Document Summarization Modules and Models
"""



from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import numpy as np
import tensorflow as tf
import random
import os

from my_flags import FLAGS
from model_utils import convert_logits_to_softmax, predict_topranked

# Special IDs
PAD_ID = 0
UNK_ID = 1
```

```
class Data:
    def __init__(self, vocab_dict, data_type):
        self.filenames = []
        self.docs = []
        self.titles = []
        self.images = []
        self.labels = []
        self.rewards = []
        self.weights = []

        self.fileindices = []

        self.data_type = data_type

        # populate the data
        self.populate_data(vocab_dict, data_type)

        # Write to files
        self.write_to_files(data_type)

    def write_prediction_summaries(self, pred_logits, modelname, session=None):
        print("Writing predictions and final summaries ...")

        # Convert to softmax logits
        pred_logits = convert_logits_to_softmax(pred_logits, session=session)
        # Save Output Logits
        np.save(FLAGS.train_dir+"/"+modelname+"."+self.data_type+"-prediction", pred_logits)

        # Writing
        pred_labels = predict_topranked(pred_logits, self.weights, self.filenames)
        self.write_predictions(modelname+"."+self.data_type, pred_logits, pred_labels)
        self.process_predictions_topranked(modelname+"."+self.data_type)
```

```
def write_predictions(self, file_prefix, np_predictions, np_labels):
    foutput = open(FLAGS.train_dir+"/"+file_prefix+".predictions", "w")
    for fileindex in self.fileindices:
        filename = self.filenames[fileindex]
        foutput.write(filename+"\n")

        sentcount = 0
        for sentpred, sentlabel in zip(np_predictions[fileindex], np_labels[fileindex]):
            one_prob = sentpred[0]
            label = sentlabel[0]

            if sentcount < len(self.weights[fileindex]):
                foutput.write(str(int(label))+"\t"+str(one_prob)+"\n")
            else:
                break

            sentcount += 1
        foutput.write("\n")
    foutput.close()
```

```
def process_predictions_topranked(self, file_prefix):
    predictiondata = open(FLAGS.train_dir+"/"+file_prefix+".predictions").read().strip().split("\n\n")
    # print len(predictiondata)

    summary_dirname = FLAGS.train_dir+"/"+file_prefix+"-summary-topranked"
    os.system("mkdir "+summary_dirname)

    for item in predictiondata:
        # print(item)

        itemdata = item.strip().split("\n")
        # print len(itemdata)

        filename = itemdata[0]
        # print filename

        # predictions file already have top three sentences marked
        final_sentids = []
        for sentid in range(len(itemdata[1:])):
            label_score = itemdata[sentid+1].split()
            if label_score[0] == "1":
                final_sentids.append(sentid)

        # Create final summary files
        fileid = filename.split("-")[-1] # cnn-fileid, dailymail-fileid
        summary_file = open(summary_dirname+"/"+fileid+".model", "w")

        # Read Sents in the document : Always use original sentences
        sent_filename = FLAGS.doc_sentence_directory + "/" + self.data_type +"/mainbody/"+fileid+".mainbody"
        docsents = open(sent_filename).readlines()

        # Top Ranked three sentences
        selected_sents = [docsents[sentid] for sentid in final_sentids if sentid < len(docsents)]
        # print(selected_sents)

        summary_file.write("".join(selected_sents)+"\n")
        summary_file.close()
```

```
    def get_batch(self, startidx, endidx):
        # This is very fast if you keep everything in Numpy

        def process_to_chop_pad(orgids, requiredsize):
            if (len(orgids) >= requiredsize):
                return orgids[:requiredsize]
            else:
                padids = [PAD_ID] * (requiredsize - len(orgids))
                return (orgids + padids)

        # Numpy dtype
        dtype = np.float16 if FLAGS.use_fp16 else np.float32

        # For train, (endidx-startidx)=FLAGS.batch_size, for others its as specified
        batch_docnames = np.empty((endidx-startidx), dtype="S60") # File ID of size "cnn-" or "dailymail-" with fileid of size 40
        batch_docs = np.empty(((endidx-startidx), (FLAGS.max_doc_length + FLAGS.max_title_length + FLAGS.max_image_length), FLAGS.max_sent_length),
dtype="int32")
        batch_label = np.empty(((endidx-startidx), FLAGS.max_doc_length, FLAGS.target_label_size), dtype=dtype) # Single best oracle, used for JP models
or accuracy estimation
        batch_weight = np.empty(((endidx-startidx), FLAGS.max_doc_length), dtype=dtype)
        batch_oracle_multiple = np.empty(((endidx-startidx), 1, FLAGS.max_doc_length, FLAGS.target_label_size), dtype=dtype)
        batch_reward_multiple = np.empty(((endidx-startidx), 1), dtype=dtype)

        batch_idx = 0
        for fileindex in self.fileindices[startidx:endidx]:
            # Document Names
            batch_docnames[batch_idx] = self.filenames[fileindex]
```

```
# Document
doc_wordids = [] # [FLAGS.max_doc_length + FLAGS.max_title_length + FLAGS.max_image_length, FLAGS.max_sent_length]
for idx in range(FLAGS.max_doc_length):
    thissent = []
    if idx < len(self.docs[fileindex]):
        thissent = self.docs[fileindex][idx][:]
    thissent = process_to_chop_pad(thissent, FLAGS.max_sent_length) # [FLAGS.max_sent_length]
    doc_wordids.append(thissent)
for idx in range(FLAGS.max_title_length):
    thissent = []
    if idx < len(self.titles[fileindex]):
        thissent = self.titles[fileindex][idx][:]
    thissent = process_to_chop_pad(thissent, FLAGS.max_sent_length) # [FLAGS.max_sent_length]
    doc_wordids.append(thissent)
for idx in range(FLAGS.max_image_length):
    thissent = []
    if idx < len(self.images[fileindex]):
        thissent = self.images[fileindex][idx][:]
    thissent = process_to_chop_pad(thissent, FLAGS.max_sent_length) # [FLAGS.max_sent_length]
    doc_wordids.append(thissent)
batch_docs[batch_idx] = np.array(doc_wordids[:], dtype="int32")

# Labels: Select the single best
labels_vecs = [[1, 0] if (item in self.labels[fileindex][0]) else [0, 1] for item in range(FLAGS.max_doc_length)]
batch_label[batch_idx] = np.array(labels_vecs[:], dtype=dtype)

# Weights
weights = process_to_chop_pad(self.weights[fileindex][:], FLAGS.max_doc_length)
batch_weight[batch_idx] = np.array(weights[:], dtype=dtype)
```

```
    # Multiple Labels and rewards
    labels_set = [] # FLAGS.num_sample_rollout, FLAGS.max_doc_length, FLAGS.target_label_size
    reward_set = [] # FLAGS.num_sample_rollout, FLAGS.max_doc_length, FLAGS.target_label_size
    for idx in range(FLAGS.num_sample_rollout):
        thislabels = []
        if idx < len(self.labels[fileindex]):
            thislabels = [[1, 0] if (item in self.labels[fileindex][idx]) else [0, 1] for item in range(FLAGS.max_doc_length)]
            reward_set.append(self.rewards[fileindex][idx])
        else:
            # Simply copy the best one
            thislabels = [[1, 0] if (item in self.labels[fileindex][0]) else [0, 1] for item in range(FLAGS.max_doc_length)]
            reward_set.append(self.rewards[fileindex][0])
        labels_set.append(thislabels)
    # Randomly Sample one oracle label
    randidx_oracle = random.randint(0, (FLAGS.num_sample_rollout-1))
    batch_oracle_multiple[batch_idx][0] = np.array(labels_set[randidx_oracle][:], dtype=dtype)
    batch_reward_multiple[batch_idx] = np.array([reward_set[randidx_oracle]], dtype=dtype)

    # increase batch count
    batch_idx += 1

return batch_docnames, batch_docs, batch_label, batch_weight, batch_oracle_multiple, batch_reward_multiple
```

```
    def shuffle_fileindices(self):
        random.shuffle(list(self.fileindices))

    def write_to_files(self, data_type):
        full_data_file_prefix = FLAGS.train_dir + "/" + FLAGS.data_mode + "." + data_type
        print("Writing data files with prefix (.filename, .doc, .title, .image, .label, .weight, .rewards): %s"%full_data_file_prefix)

        ffilenames = open(full_data_file_prefix+".filename", "w")
        fdoc = open(full_data_file_prefix+".doc", "w")
        ftitle = open(full_data_file_prefix+".title", "w")
        fimage = open(full_data_file_prefix+".image", "w")
        flabel = open(full_data_file_prefix+".label", "w")
        fweight = open(full_data_file_prefix+".weight", "w")
        freward = open(full_data_file_prefix+".reward", "w")

        for filename, doc, title, image, label, weight, reward in zip(self.filenames, self.docs, self.titles, self.images, self.labels, self.weights,
self.rewards):
            ffilenames.write(filename+"\n")
            fdoc.write("\n".join([" ".join([str(item) for item in itemlist]) for itemlist in doc])+"\n\n")
            ftitle.write("\n".join([" ".join([str(item) for item in itemlist]) for itemlist in title])+"\n\n")
            fimage.write("\n".join([" ".join([str(item) for item in itemlist]) for itemlist in image])+"\n\n")
            flabel.write("\n".join([" ".join([str(item) for item in itemlist]) for itemlist in label])+"\n\n")
            fweight.write(" ".join([str(item) for item in weight])+"\n")
            freward.write(" ".join([str(item) for item in reward])+"\n")

        ffilenames.close()
        fdoc.close()
        ftitle.close()
        fimage.close()
        flabel.close()
        fweight.close()
        freward.close()
```

```
def populate_data(self, vocab_dict, data_type):

    full_data_file_prefix = FLAGS.preprocessed_data_directory + "/" + FLAGS.data_mode + "." + data_type
    print("Data file prefix (.doc, .title, .image, .label.multipleoracle): %s"%full_data_file_prefix)

    # Process doc, title, image, label
    doc_data_list = open(full_data_file_prefix+".doc").read().strip().split("\n\n")
    title_data_list = open(full_data_file_prefix+".title").read().strip().split("\n\n")
    image_data_list = open(full_data_file_prefix+".image").read().strip().split("\n\n")
    label_data_list = open(full_data_file_prefix+".label.multipleoracle").read().strip().split("\n\n")

    print("Data sizes: %d %d %d %d"%(len(doc_data_list), len(title_data_list), len(image_data_list), len(label_data_list)))

    print("Reading data (no padding to save memory) ...")
    doccount = 0
    for doc_data, title_data, image_data, label_data in zip(doc_data_list, title_data_list, image_data_list, label_data_list):

        doc_lines = doc_data.strip().split("\n")
        title_lines = title_data.strip().split("\n")
        image_lines = image_data.strip().split("\n")
        label_lines = label_data.strip().split("\n")

        filename = doc_lines[0].strip()

        if ((filename == title_lines[0].strip()) and (filename == image_lines[0].strip()) and (filename == label_lines[0].strip())):
            # Put filename
            self.filenames.append(filename)

            # Doc
            thisdoc = []
            for line in doc_lines[1:FLAGS.max_doc_length+1]:
                thissent = [int(item) for item in line.strip().split()]
                thisdoc.append(thissent)
            self.docs.append(thisdoc)

            # Title
            thistitle = []
```

```
            for line in title_lines[1:FLAGS.max_title_length+1]:
                thissent = [int(item) for item in line.strip().split()]
                thistitle.append(thissent)
            self.titles.append(thistitle)

            # Image
            thisimage = []
            for line in image_lines[1:FLAGS.max_image_length+1]:
                thissent = [int(item) for item in line.strip().split()]
                thisimage.append(thissent)
            self.images.append(thisimage)

            # Weights
            originaldoclen = int(label_lines[1].strip())
            thisweight = [1 for item in range(originaldoclen)][:FLAGS.max_doc_length]
            self.weights.append(thisweight)

            # Labels (multiple oracles and preestimated rewards)
            thislabel = []
            thisreward = []
            for line in label_lines[2:FLAGS.num_sample_rollout+2]:
                thislabel.append([int(item) for item in line.split()[:-1]])
                thisreward.append(float(line.split()[-1]))
            self.labels.append(thislabel)
            self.rewards.append(thisreward)

        else:
            print("Some problem with %s.* files. Exiting!"%full_data_file_prefix)
            exit(0)

        if doccount%10000==0:
            print("%d ..."%doccount)
        doccount += 1

    # Set Fileindices
    self.fileindices = range(len(self.filenames))
```

```
class DataProcessor:
    def prepare_news_data(self, vocab_dict, data_type="training"):
        data = Data(vocab_dict, data_type)
        return data

    def prepare_vocab_embeddingdict(self):
        # Numpy dtype
        dtype = np.float16 if FLAGS.use_fp16 else np.float32

        vocab_dict = {}
        word_embedding_array = []

        # Add padding
        vocab_dict["_PAD"] = PAD_ID
        # Add UNK
        vocab_dict["_UNK"] = UNK_ID
```

```
# Read word embedding file
wordembed_filename = FLAGS.pretrained_wordembedding
print("Reading pretrained word embeddings file: %s"%wordembed_filename)

embed_line = ""
linecount = 0
with open(wordembed_filename, "r",encoding='UTF-8') as fembedd:
    for line in fembedd:
        if linecount == 0:
            vocabsize = int(line.split()[0])
            # Initiate fixed size empty array
            word_embedding_array = np.empty((vocabsize, FLAGS.wordembed_size), dtype=dtype)
        else:
            linedata = line.split()
            vocab_dict[linedata[0]] = linecount + 1
            embeddata = [float(item) for item in linedata[1:]][0:FLAGS.wordembed_size]
            word_embedding_array[linecount-1] = np.array(embeddata, dtype=dtype)

        if linecount%100000 == 0:
            print(str(linecount)+" ...")
        linecount += 1
print("Read pretrained embeddings: %s"%str(word_embedding_array.shape))

print("Size of vocab: %d (_PAD:0, _UNK:1)"%len(vocab_dict))
vocabfilename = FLAGS.train_dir+"\\vocab.txt"
print("Writing vocab file: %s"%vocabfilename)

foutput = open(vocabfilename,"w",encoding='UTF-8')
vocab_list = [(vocab_dict[key], key) for key in vocab_dict.keys()]
vocab_list.sort()
vocab_list = [item[1] for item in vocab_list]
foutput.write("\n".join(vocab_list)+"\n")
foutput.close()
return vocab_dict, word_embedding_array
```