```
# Author: Shashi Narayan
# Date: September 2016
# Project: Document Summarization
# H2020 Summa Project
# Comments: Jan 2017
# Improved for Reinforcement Learning
#####################################

"""
Document Summarization System
"""

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

import math
import os
import random
import sys
import time

import numpy as np
import tensorflow as tf

from reward_utils import Reward_Generator
from data_utils import DataProcessor
from my_flags import FLAGS
from my_model import MY_Model
```

```
######################## Batch Testing a model on some dataset #############

def batch_predict_with_a_model(data, model, session=None):

  data_logits = []
  data_labels = []
  data_weights = []

  step = 1
  while (step * FLAGS.batch_size) <= len(data.fileindices):
    # Get batch data as Numpy Arrays : Without shuffling
    batch_docnames, batch_docs, batch_label, batch_weight, batch_oracle_multiple, batch_reward_multiple = data.get_batch(((step-1)*FLAGS.batch_size),
(step * FLAGS.batch_size))
    batch_logits = session.run(model.logits, feed_dict={model.document_placeholder: batch_docs})

    data_logits.append(batch_logits)
    data_labels.append(batch_label)
    data_weights.append(batch_weight)

    # Increase step
    step += 1

  # Check if any data left
  if (len(data.fileindices) > ((step-1)*FLAGS.batch_size)):
    # Get last batch as Numpy Arrays
    batch_docnames, batch_docs, batch_label, batch_weight, batch_oracle_multiple, batch_reward_multiple = data.get_batch(((step-1)*FLAGS.batch_size),
len(data.fileindices))
    batch_logits = session.run(model.logits, feed_dict={model.document_placeholder: batch_docs})

    data_logits.append(batch_logits)
    data_labels.append(batch_label)
    data_weights.append(batch_weight)
    # print(data_logits)



  # Convert list to tensors
```

```
  data_logits = tf.concat(0, data_logits)
  data_lables = tf.concat(0, data_labels)
  data_weights = tf.concat(0, data_weights)
  # print(data_logits,data_lables,data_weights)
  return data_logits, data_lables, data_weights

######################## Training Mode ############################

def train():
  """
  Training Mode: Create a new model and train the network
  """

  # Training: use the tf default graph
  with tf.Graph().as_default() and tf.device(FLAGS.use_gpu):

    config = tf.ConfigProto(allow_soft_placement = True)

    # Start a session
    with tf.Session(config = config) as sess:

      ### Prepare data for training
      print("Prepare vocab dict and read pretrained word embeddings ...")
      vocab_dict, word_embedding_array = DataProcessor().prepare_vocab_embeddingdict()
      # vocab_dict contains _PAD and _UNK but not word_embedding_array

      print("Prepare training data ...")
      train_data = DataProcessor().prepare_news_data(vocab_dict, data_type="training")

      print("Prepare validation data ...")
      validation_data = DataProcessor().prepare_news_data(vocab_dict, data_type="validation")

      print("Prepare ROUGE reward generator ...")
      rouge_generator = Reward_Generator()


      # Create Model with various operations
```

```
model = MY_Model(sess, len(vocab_dict)-2)

# Start training with some pretrained model
start_epoch = 1
# selected_modelpath = FLAGS.train_dir+"/model.ckpt.epoch-"+str(start_epoch-1)
# if not (os.path.isfile(selected_modelpath)):
#   print("Model not found in checkpoint folder.")
#   exit(0)
# # Reload saved model and test
# print("Reading model parameters from %s" % selected_modelpath)
# model.saver.restore(sess, selected_modelpath)
# print("Model loaded.")

# Initialize word embedding before training
print("Initialize word embedding vocabulary with pretrained embeddings ...")
sess.run(model.vocab_embed_variable.assign(word_embedding_array))

########### Start (No Mixer) Training : Reinforcement learning ################
# Reward aware training as part of Reward weighted CE ,
# No Curriculam learning: No annealing, consider full document like in MRT
# Multiple Samples (include gold sample),  No future reward, Similar to MRT
# During training does not use PYROUGE to avoid multiple file rewritings
# Approximate MRT with multiple pre-estimated oracle samples
# June 2017: Use Single sample from multiple oracles
###############################################################################

print("Start Reinforcement Training (single rollout from largest prob mass) ...")
```

```
    for epoch in range(start_epoch, FLAGS.train_epoch_wce + 1):
      print("MRT: Epoch "+str(epoch))

      print("MRT: Epoch "+str(epoch)+" : Reshuffle training document indices")
      train_data.shuffle_fileindices()

      print("MRT: Epoch "+str(epoch)+" : Restore Rouge Dict")
      rouge_generator.restore_rouge_dict()

      # Start Batch Training
      step = 1
      while (step * FLAGS.batch_size) <= len(train_data.fileindices):
        # Get batch data as Numpy Arrays
        batch_docnames, batch_docs, batch_label, batch_weight, batch_oracle_multiple, batch_reward_multiple = train_data.get_batch(((step-
1)*FLAGS.batch_size),
                                                                                                                                    (step * FLAGS.batch_size))
        # print(batch_docnames)
        # print(batch_label[0])
        # print(batch_weight[0])
        # print(batch_oracle_multiple[0])
        # print(batch_reward_multiple[0])
        # exit(0)

        # Print the progress
        if (step % FLAGS.training_checkpoint)  == 0:

          ce_loss_val, ce_loss_sum, acc_val, acc_sum = sess.run([model.rewardweighted_cross_entropy_loss_multisample,
model.rewardweighted_ce_multisample_loss_summary,
                                                                 model.accuracy, model.taccuracy_summary],
                                                                 feed_dict={model.document_placeholder: batch_docs,
                                                                            model.predicted_multisample_label_placeholder: batch_oracle_multiple,
                                                                            model.actual_reward_multisample_placeholder: batch_reward_multiple,
                                                                            model.label_placeholder: batch_label,
                                                                            model.weight_placeholder: batch_weight})

          # Print Summary to Tensor Board
```

```
            model.summary_writer.add_summary(ce_loss_sum, ((epoch-1)*len(train_data.fileindices)+ step*FLAGS.batch_size))
            model.summary_writer.add_summary(acc_sum, ((epoch-1)*len(train_data.fileindices)+step*FLAGS.batch_size))

            print("MRT: Epoch "+str(epoch)+" : Covered " + str(step*FLAGS.batch_size)+"/"+str(len(train_data.fileindices)) +
                  " : Minibatch Reward Weighted Multisample CE Loss= {:.6f}".format(ce_loss_val) + " : Minibatch training accuracy=
{:.6f}".format(acc_val))

          # Run optimizer: optimize policy network
          sess.run([model.train_op_policynet_expreward], feed_dict={model.document_placeholder: batch_docs,
                                                                    model.predicted_multisample_label_placeholder: batch_oracle_multiple,
                                                                    model.actual_reward_multisample_placeholder: batch_reward_multiple,
                                                                    model.weight_placeholder: batch_weight})

          # Increase step
          step += 1

          # if step == 20:
          #   break

        # Save Model
        print("MRT: Epoch "+str(epoch)+" : Saving model after epoch completion")
        checkpoint_path = os.path.join(FLAGS.train_dir, "model.ckpt.epoch-"+str(epoch))
        model.saver.save(sess, checkpoint_path)

        # Backup Rouge Dict
        print("MRT: Epoch "+str(epoch)+" : Saving rouge dictionary")
        rouge_generator.save_rouge_dict()

        # Performance on the validation set
        print("MRT: Epoch "+str(epoch)+" : Performance on the validation data")




        # Get Predictions: Prohibit the use of gold labels
        validation_logits, validation_labels, validation_weights = batch_predict_with_a_model(validation_data, model, session=sess)
```

```
        # Validation Accuracy and Prediction
        validation_acc, validation_sum = sess.run([model.final_accuracy, model.vaccuracy_summary], feed_dict={model.logits_placeholder:
validation_logits.eval(session=sess),
                                                                                                               model.label_placeholder:
validation_labels.eval(session=sess),
                                                                                                               model.weight_placeholder:
validation_weights.eval(session=sess)})
        # Print Validation Summary
        model.summary_writer.add_summary(validation_sum, (epoch*len(train_data.fileindices)))

        print("MRT: Epoch "+str(epoch)+" : Validation ("+str(len(validation_data.fileindices))+") accuracy= {:.6f}".format(validation_acc))
        # Writing validation predictions and final summaries
        print("MRT: Epoch "+str(epoch)+" : Writing final validation summaries")
        validation_data.write_prediction_summaries(validation_logits, "model.ckpt.epoch-"+str(epoch), session=sess)
        # Extimate Rouge Scores
        rouge_score = rouge_generator.get_full_rouge(FLAGS.train_dir+"/model.ckpt.epoch-"+str(epoch)+".validation-summary-topranked", "validation")
        print("MRT: Epoch "+str(epoch)+" : Validation ("+str(len(validation_data.fileindices))+") rouge= {:.6f}".format(rouge_score))

        # break

    print("Optimization Finished!")




# ######################## Test Mode ############################
```

```
def test():
  """
  Test Mode: Loads an existing model and test it on the test set
  """

  # Training: use the tf default graph

  with tf.Graph().as_default() and tf.device(FLAGS.use_gpu):

    config = tf.ConfigProto(allow_soft_placement = True)

    # Start a session
    with tf.Session(config = config) as sess:

      ### Prepare data for training
      print("Prepare vocab dict and read pretrained word embeddings ...")
      vocab_dict, word_embedding_array = DataProcessor().prepare_vocab_embeddingdict()
      # vocab_dict contains _PAD and _UNK but not word_embedding_array

      print("Prepare test data ...")
      test_data = DataProcessor().prepare_news_data(vocab_dict, data_type="test")

      # Create Model with various operations
      model = MY_Model(sess, len(vocab_dict)-2)

      # # Initialize word embedding before training
      # print("Initialize word embedding vocabulary with pretrained embeddings ...")
      # sess.run(model.vocab_embed_variable.assign(word_embedding_array))

      # Select the model
      if (os.path.isfile(FLAGS.train_dir+"/model.ckpt.epoch-"+str(FLAGS.model_to_load))):
        selected_modelpath = FLAGS.train_dir+"/model.ckpt.epoch-"+str(FLAGS.model_to_load)
      else:
        print("Model not found in checkpoint folder.")
        exit(0)

      # Reload saved model and test
```

```
        print("Reading model parameters from %s" % selected_modelpath)
        model.saver.restore(sess, selected_modelpath)
        print("Model loaded.")

        # Initialize word embedding before training
        print("Initialize word embedding vocabulary with pretrained embeddings ...")
        sess.run(model.vocab_embed_variable.assign(word_embedding_array))

        # Test Accuracy and Prediction
        print("Performance on the test data:")
        FLAGS.authorise_gold_label = False
        test_logits, test_labels, test_weights = batch_predict_with_a_model(test_data, model, session=sess)
        test_acc = sess.run(model.final_accuracy, feed_dict={model.logits_placeholder: test_logits.eval(session=sess),
                                                             model.label_placeholder: test_labels.eval(session=sess),
                                                             model.weight_placeholder: test_weights.eval(session=sess)})
        # Print Test Summary
        print("Test ("+str(len(test_data.fileindices))+") accuracy= {:.6f}".format(test_acc))
        # Writing test predictions and final summaries
        test_data.write_prediction_summaries(test_logits, "model.ckpt.epoch-"+str(FLAGS.model_to_load), session=sess)

######################## Main Function ##########################

def main(_):
  if FLAGS.exp_mode == "train":
    train()
  else:
    test()

if __name__ == "__main__":
  tf.app.run()
```