# coding=utf-8

# Copyright 2018 Google LLC & Hwalsuk Lee.

#

# Licensed under the Apache License, Version 2.0 (the "License");

# you may not use this file except in compliance with the License.

# You may obtain a copy of the License at

#

# http://www.apache.org/licenses/LICENSE-2.0

#

# Unless required by applicable law or agreed to in writing, software

# distributed under the License is distributed on an "AS IS" BASIS,

# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

# See the License for the specific language governing permissions and

# limitations under the License.

"""Discriminator accuracy.

Computes the discrimionator's accuracy on (a subset) of the training dataset,

test dataset, and a generated data set. The score is averaged over several

multiple generated data sets and subsets of the training data.

"""

from \_\_future\_\_ import absolute\_import

from \_\_future\_\_ import division

from \_\_future\_\_ import print\_function

from absl import logging

from compare\_gan import datasets

from compare\_gan import eval\_utils

from compare\_gan.metrics import eval\_task

import numpy as np

class AccuracyTask(eval\_task.EvalTask):

"""Evaluation Task for computing and reporting accuracy."""

def metric\_list(self):

return frozenset([

"train\_accuracy", "test\_accuracy", "fake\_accuracy", "train\_d\_loss",

"test\_d\_loss"

])

def run\_in\_session(self, options, sess, gan, real\_images):

del options

return compute\_accuracy\_loss(sess, gan, real\_images)

def compute\_accuracy\_loss(sess,

gan,

test\_images,

max\_train\_examples=50000,

num\_repeat=5):

"""Compute discriminator's accuracy and loss on a given dataset.

Args:

sess: Tf.Session object.

gan: Any AbstractGAN instance.

test\_images: numpy array with test images.

max\_train\_examples: How many "train" examples to get from the dataset.

In each round, some of them will be randomly selected

to evaluate train set accuracy.

num\_repeat: How many times to repreat the computation.

The mean of all the results is reported.

Returns:

Dict[Text, float] with all the computed scores.

Raises:

ValueError: If the number of test\_images is greater than the number of

training images returned by the dataset.

"""

logging.info("Evaluating training and test accuracy...")

train\_images = eval\_utils.get\_real\_images(

dataset=datasets.get\_dataset(),

num\_examples=max\_train\_examples,

split="train",

failure\_on\_insufficient\_examples=False)

if train\_images.shape[0] < test\_images.shape[0]:

raise ValueError("num\_train %d must be larger than num\_test %d." %

(train\_images.shape[0], test\_images.shape[0]))

num\_batches = int(np.floor(test\_images.shape[0] / gan.batch\_size))

if num\_batches \* gan.batch\_size < test\_images.shape[0]:

logging.error("Ignoring the last batch with %d samples / %d epoch size.",

test\_images.shape[0] - num\_batches \* gan.batch\_size,

gan.batch\_size)

ret = {

"train\_accuracy": [],

"test\_accuracy": [],

"fake\_accuracy": [],

"train\_d\_loss": [],

"test\_d\_loss": []

}

for \_ in range(num\_repeat):

idx = np.random.choice(train\_images.shape[0], test\_images.shape[0])

bs = gan.batch\_size

train\_subset = [train\_images[i] for i in idx]

train\_predictions, test\_predictions, fake\_predictions = [], [], []

train\_d\_losses, test\_d\_losses = [], []

for i in range(num\_batches):

z\_sample = gan.z\_generator(gan.batch\_size, gan.z\_dim)

start\_idx = i \* bs

end\_idx = start\_idx + bs

test\_batch = test\_images[start\_idx : end\_idx]

train\_batch = train\_subset[start\_idx : end\_idx]

test\_prediction, test\_d\_loss, fake\_images = sess.run(

[gan.discriminator\_output, gan.d\_loss, gan.fake\_images],

feed\_dict={

gan.inputs: test\_batch, gan.z: z\_sample

})

train\_prediction, train\_d\_loss = sess.run(

[gan.discriminator\_output, gan.d\_loss],

feed\_dict={

gan.inputs: train\_batch,

gan.z: z\_sample

})

fake\_prediction = sess.run(

gan.discriminator\_output,

feed\_dict={gan.inputs: fake\_images})[0]

train\_predictions.append(train\_prediction[0])

test\_predictions.append(test\_prediction[0])

fake\_predictions.append(fake\_prediction)

train\_d\_losses.append(train\_d\_loss)

test\_d\_losses.append(test\_d\_loss)

train\_predictions = [x >= 0.5 for x in train\_predictions]

test\_predictions = [x >= 0.5 for x in test\_predictions]

fake\_predictions = [x < 0.5 for x in fake\_predictions]

ret["train\_accuracy"].append(np.array(train\_predictions).mean())

ret["test\_accuracy"].append(np.array(test\_predictions).mean())

ret["fake\_accuracy"].append(np.array(fake\_predictions).mean())

ret["train\_d\_loss"].append(np.mean(train\_d\_losses))

ret["test\_d\_loss"].append(np.mean(test\_d\_losses))

for key in ret:

ret[key] = np.mean(ret[key])

return ret