```python
'''
An example of distribution approximation using Generative Adversarial Networks in TensorFlow.
Based on the blog post by Eric Jang: http://blog.evjang.com/2016/06/generative-adversarial-nets-in.html,
and of course the original GAN paper by Ian Goodfellow et. al.: https://arxiv.org/abs/1406.2661.
The minibatch discrimination technique is taken from Tim Salimans et. al.: https://arxiv.org/abs/1606.03498.
'''
from __future__ import absolute_import
from __future__ import print_function
from __future__ import unicode_literals
from __future__ import division

import argparse
import numpy as np
from scipy.stats import norm
import tensorflow as tf
import matplotlib.pyplot as plt
from matplotlib import animation
# import seaborn as sns

# sns.set(color_codes=True)

seed = 42
np.random.seed(seed)
tf.set_random_seed(seed)


class DataDistribution(object):
    def __init__(self):
        self.mu = 4
        self.sigma = 0.5

    def sample(self, N):
        samples = np.random.normal(self.mu, self.sigma, N)
        samples.sort()
        return samples


class GeneratorDistribution(object):
    def __init__(self, range):
        self.range = range

    def sample(self, N):
        return np.linspace(-self.range, self.range, N) + W
            np.random.random(N) * 0.01


def linear(input, output_dim, scope=None, stddev=1.0):
    norm = tf.random_normal_initializer(stddev=stddev)
    const = tf.constant_initializer(0.0)
    with tf.variable_scope(scope or 'linear'):
        w = tf.get_variable('w', [input.get_shape()[1], output_dim], initializer=norm)
```

42으로 바꿈, GAN algorithm 자체가 random seed에 민감한듯.
42로 한 경우 수렴하지 않음.

normal distribution에서 N개의 sample을 뽑고, 이를 그래프로 정리해 return.

-range ~ range 사이를 N개로 나누고, random noise를 더해서 return.

```
52          b = tf.get_variable('b', [output_dim], initializer=const)
53          return tf.matmul(input, w) + b
54
55
56  def generator(input, h_dim):
57      h0 = tf.nn.softplus(linear(input, h_dim, 'g0'))
58      h1 = linear(h0, 1, 'g1')
59      return h1
60
61
62  def discriminator(input, h_dim, minibatch_layer=True):
63      h0 = tf.tanh(linear(input, h_dim * 2, 'd0'))
64      h1 = tf.tanh(linear(h0, h_dim * 2, 'd1'))
65
66      # without the minibatch layer, the discriminator needs an additional layer
67      # to have enough capacity to separate the two distributions correctly
68      if minibatch_layer:
69          h2 = minibatch(h1)
70      else:
71          h2 = tf.tanh(linear(h1, h_dim * 2, scope='d2'))
72
73      h3 = tf.sigmoid(linear(h2, 1, scope='d3'))
74      return h3
75
76
77  def minibatch(input, num_kernels=5, kernel_dim=3):
78      x = linear(input, num_kernels * kernel_dim, scope='minibatch', stddev=0.02)
79      activation = tf.reshape(x, (-1, num_kernels, kernel_dim))
80      diffs = tf.expand_dims(activation, 3) - tf.expand_dims(tf.transpose(activation, [1, 2, 0])
    , 0)
81      abs_diffs = tf.reduce_sum(tf.abs(diffs), 2)
82      minibatch_features = tf.reduce_sum(tf.exp(-abs_diffs), 2)
83      return tf.concat(1, [input, minibatch_features])
84
85
86  def optimizer(loss, var_list, initial_learning_rate):
87      decay = 0.95
88      num_decay_steps = 150
89      batch = tf.Variable(0)
90      learning_rate = tf.train.exponential_decay(
91          initial_learning_rate,
92          batch,
93          num_decay_steps,
94          decay,
95          staircase=True
96      )
97      optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(
98          loss,
99          global_step=batch,
100         var_list=var_list
101     )
102     return optimizer
103
104
```

```python
class GAN(object):
    def __init__(self, data, gen, num_steps, batch_size, minibatch, log_every, anim_path):
        self.data = data
        self.gen = gen
        self.num_steps = num_steps
        self.batch_size = batch_size
        self.minibatch = minibatch
        self.log_every = log_every
        self.mlp_hidden_size = 4
        self.anim_path = anim_path
        self.anim_frames = []

        # can use a higher learning rate when not using the minibatch layer
        if self.minibatch:
            self.learning_rate = 0.005
        else:
            self.learning_rate = 0.03

        self._create_model()

    def _create_model(self):
        # In order to make sure that the discriminator is providing useful gradient
        # information to the generator from the start, we're going to pretrain the
        # discriminator using a maximum likelihood objective. We define the network
        # for this pretraining step scoped as D_pre.
        with tf.variable_scope('D_pre'):
            self.pre_input = tf.placeholder(tf.float32, shape=(self.batch_size, 1))
            self.pre_labels = tf.placeholder(tf.float32, shape=(self.batch_size, 1))
            D_pre = discriminator(self.pre_input, self.mlp_hidden_size, self.minibatch)
            self.pre_loss = tf.reduce_mean(tf.square(D_pre - self.pre_labels))
            self.pre_opt = optimizer(self.pre_loss, None, self.learning_rate)

        # This defines the generator network - it takes samples from a noise
        # distribution as input, and passes them through an MLP.
        with tf.variable_scope('Gen'):
            self.z = tf.placeholder(tf.float32, shape=(self.batch_size, 1))
            self.G = generator(self.z, self.mlp_hidden_size)

        # The discriminator tries to tell the difference between samples from the
        # true data distribution (self.x) and the generated samples (self.z).
        #
        # Here we create two copies of the discriminator network (that share parameters),
        # as you cannot use the same network with different inputs in TensorFlow.
        with tf.variable_scope('Disc') as scope:
            self.x = tf.placeholder(tf.float32, shape=(self.batch_size, 1))
            self.D1 = discriminator(self.x, self.mlp_hidden_size, self.minibatch)
            scope.reuse_variables()
            self.D2 = discriminator(self.G, self.mlp_hidden_size, self.minibatch)

        # Define the loss for discriminator and generator networks (see the original
        # paper for details), and create optimizers for both
        self.loss_d = tf.reduce_mean(-tf.log(self.D1) - tf.log(1 - self.D2))
        self.loss_g = tf.reduce_mean(-tf.log(self.D2))

```

discriminator가 학습초기에 generator에게 충분한 gradient를 주게하고 discriminator를 pretrain 할 network를 D_pre라함

→ real data에 대한 discriminator network.

→ generator가 만든 data에 대한 discriminator network.

두 network는 parameter를 공유한다.

즉, 같은 network에 2개의 input을 각각 넣어 결과를 보는것

⇒ D(x)와 D(G(z))

원래는 log(1-self.D2)가 맞지만 논문에서 안정성등이 초기의 saturation을 방지하기위해 log(self.D2) 사용.

```
159         self.d_pre_params = tf.get_collection(tf.GraphKeys.TRAINABLE_VARIABLES, scope='D_pre')
160         self.d_params = tf.get_collection(tf.GraphKeys.TRAINABLE_VARIABLES, scope='Disc')
161         self.g_params = tf.get_collection(tf.GraphKeys.TRAINABLE_VARIABLES, scope='Gen')
162
163         self.opt_d = optimizer(self.loss_d, self.d_params, self.learning_rate)
164         self.opt_g = optimizer(self.loss_g, self.g_params, self.learning_rate)
165
166     def train(self):
167         with tf.Session() as session:
168             tf.global_variables_initializer().run()
169
170             # pretraining discriminator
171             num_pretrain_steps = 1000
172             for step in range(num_pretrain_steps):
173                 d = (np.random.random(self.batch_size) - 0.5) * 10.0
174                 labels = norm.pdf(d, loc=self.data.mu, scale=self.data.sigma)
175                 pretrain_loss, _ = session.run([self.pre_loss, self.pre_opt], {
176                     self.pre_input: np.reshape(d, (self.batch_size, 1)),
177                     self.pre_labels: np.reshape(labels, (self.batch_size, 1))
178                 })
179             self.weightsD = session.run(self.d_pre_params)
180
181             # copy weights from pre-training over to new D network
182             for i, v in enumerate(self.d_params):
183                 session.run(v.assign(self.weightsD[i]))
184
185             for step in range(self.num_steps):
186                 # update discriminator
187                 x = self.data.sample(self.batch_size)
188                 z = self.gen.sample(self.batch_size)
189                 loss_d, _ = session.run([self.loss_d, self.opt_d], {
190                     self.x: np.reshape(x, (self.batch_size, 1)),
191                     self.z: np.reshape(z, (self.batch_size, 1))
192                 })
193
194                 # update generator
195                 z = self.gen.sample(self.batch_size)
196                 loss_g, _ = session.run([self.loss_g, self.opt_g], {
197                     self.z: np.reshape(z, (self.batch_size, 1))
198                 })
199
200                 if step % self.log_every == 0:
201                     print('{}: {}Wt{}'.format(step, loss_d, loss_g))
202
203                 if self.anim_path:
204                     self.anim_frames.append(self._samples(session))
205
206             if self.anim_path:
207                 self._save_animation()
208             else:
209                 self._plot_distributions(session)
210
211     def _samples(self, session, num_points=10000, num_bins=100):
212         '''
```

Pretrain을 위한 Data set.

각각의 data 에 대한 실제 probability density function에서의 probability 값을 label로 사용

learning한 parameter를 불러옴.

```
        Return a tuple (db, pd, pg), where db is the current decision
        boundary, pd is a histogram of samples from the data distribution,
        and pg is a histogram of generated samples.
        '''
        xs = np.linspace(-self.gen.range, self.gen.range, num_points)
        bins = np.linspace(-self.gen.range, self.gen.range, num_bins)

        # decision boundary
        db = np.zeros((num_points, 1))
        for i in range(num_points // self.batch_size):
            db[self.batch_size * i:self.batch_size * (i + 1)] = session.run(self.D1, {
                self.x: np.reshape(
                    xs[self.batch_size * i:self.batch_size * (i + 1)],
                    (self.batch_size, 1)
                )
            })

        # data distribution
        d = self.data.sample(num_points)
        pd, _ = np.histogram(d, bins=bins, density=True)

        # generated samples
        zs = np.linspace(-self.gen.range, self.gen.range, num_points)
        g = np.zeros((num_points, 1))
        for i in range(num_points // self.batch_size):
            g[self.batch_size * i:self.batch_size * (i + 1)] = session.run(self.G, {
                self.z: np.reshape(
                    zs[self.batch_size * i:self.batch_size * (i + 1)],
                    (self.batch_size, 1)
                )
            })
        pg, _ = np.histogram(g, bins=bins, density=True)

        return db, pd, pg

    def _plot_distributions(self, session):
        db, pd, pg = self._samples(session)
        db_x = np.linspace(-self.gen.range, self.gen.range, len(db))
        p_x = np.linspace(-self.gen.range, self.gen.range, len(pd))
        f, ax = plt.subplots(1)
        ax.plot(db_x, db, label='decision boundary')
        ax.set_ylim(0, 1)
        plt.plot(p_x, pd, label='real data')
        plt.plot(p_x, pg, label='generated data')
        plt.title('1D Generative Adversarial Network')
        plt.xlabel('Data values')
        plt.ylabel('Probability density')
        plt.legend()
        plt.show()

    def _save_animation(self):
        f, ax = plt.subplots(figsize=(6, 4))
        f.suptitle('1D Generative Adversarial Network', fontsize=15)
        plt.xlabel('Data values')
```

```
267         plt.ylabel('Probability density')
268         ax.set_xlim(-6, 6)
269         ax.set_ylim(0, 1.4)
270         line_db, = ax.plot([], [], label='decision boundary')
271         line_pd, = ax.plot([], [], label='real data')
272         line_pg, = ax.plot([], [], label='generated data')
273         frame_number = ax.text(
274             0.02,
275             0.95,
276             '',
277             horizontalalignment='left',
278             verticalalignment='top',
279             transform=ax.transAxes
280         )
281         ax.legend()
282
283         db, pd, _ = self.anim_frames[0]
284         db_x = np.linspace(-self.gen.range, self.gen.range, len(db))
285         p_x = np.linspace(-self.gen.range, self.gen.range, len(pd))
286
287         def init():
288             line_db.set_data([], [])
289             line_pd.set_data([], [])
290             line_pg.set_data([], [])
291             frame_number.set_text('')
292             return (line_db, line_pd, line_pg, frame_number)
293
294         def animate(i):
295             frame_number.set_text(
296                 'Frame: {}/{}'.format(i, len(self.anim_frames))
297             )
298             db, pd, pg = self.anim_frames[i]
299             line_db.set_data(db_x, db)
300             line_pd.set_data(p_x, pd)
301             line_pg.set_data(p_x, pg)
302             return (line_db, line_pd, line_pg, frame_number)
303
304         anim = animation.FuncAnimation(
305             f,
306             animate,
307             init_func=init,
308             frames=len(self.anim_frames),
309             blit=True
310         )
311         anim.save(self.anim_path, fps=30, extra_args=['-vcodec', 'libx264'])
312         plt.show() - 추가, 최종적로 생성되는 figure에 animation을 보이줌.
313
314 def main(args):
315     model = GAN(
316         DataDistribution(),
317         GeneratorDistribution(range=8),
318         args.num_steps,
319         args.batch_size,
320         args.minibatch,
```

'GAN_intro.mp4' → 파일 확장자명을 꼭 입력해줘야 하는듯.

이 코드를 사용해 동영상을 재생하기 위해 FFMPEG를 설치해야함.

```
321         args.log_every,
322         args.anim
323     )
324     model.train()
325
326
327 def parse_args():
328     parser = argparse.ArgumentParser()
329     parser.add_argument('--num-steps', type=int, default=1200,
330                         help='the number of training steps to take')
331     parser.add_argument('--batch-size', type=int, default=12,
332                         help='the batch size')
333     parser.add_argument('--minibatch', type=bool, default=False,
334                         help='use minibatch discrimination')
335     parser.add_argument('--log-every', type=int, default=10,
336                         help='print loss after this many steps')
337     parser.add_argument('--anim', type=str, default=None,'test.mp4' → test.mp4 로 default로
338                         help='name of the output animation file (default: none)')
339     return parser.parse_args()                                    test.mp4
340
341
342 if __name__ == '__main__':
343     main(parse_args())
```