dlnd_face_generation

August 4, 2018

1 Face Generation

In this project, you'll use generative adversarial networks to generate new images of faces. ### Get the Data You'll be using two datasets in this project: - MNIST - CelebA

Since the celebA dataset is complex and you're doing GANs in a project for the first time, we want you to test your neural network on MNIST before CelebA. Running the GANs on MNIST will allow you to see how well your model trains sooner.

If you're using FloydHub, set `data_dir` to "/input" and use the FloydHub data ID "R5KrjnANiKVhLWAkpXhNBe".

In [13]:
```
data_dir = '/data'
!pip install matplotlib==2.0.2
# FloydHub - Use with data ID "R5KrjnANiKVhLWAkpXhNBe"
#data_dir = '/input'


"""
DON'T MODIFY ANYTHING IN THIS CELL
"""
import helper

helper.download_extract('mnist', data_dir)
helper.download_extract('celeba', data_dir)
```

```
Requirement already satisfied: matplotlib==2.0.2 in /opt/conda/lib/python3.6/site-packages
Requirement already satisfied: pytz in /opt/conda/lib/python3.6/site-packages (from matplotlib==
Requirement already satisfied: pyparsing!=2.0.0,!=2.0.4,!=2.1.2,!=2.1.6,>=1.5.6 in /opt/conda/li
Requirement already satisfied: python-dateutil in /opt/conda/lib/python3.6/site-packages (from m
Requirement already satisfied: cycler>=0.10 in /opt/conda/lib/python3.6/site-packages/cycler-0.1
Requirement already satisfied: numpy>=1.7.1 in /opt/conda/lib/python3.6/site-packages (from matp
Requirement already satisfied: six>=1.10 in /opt/conda/lib/python3.6/site-packages (from matplot
You are using pip version 9.0.1, however version 18.0 is available.You should consider upgrading
Found mnist Data
Found celeba Data
```

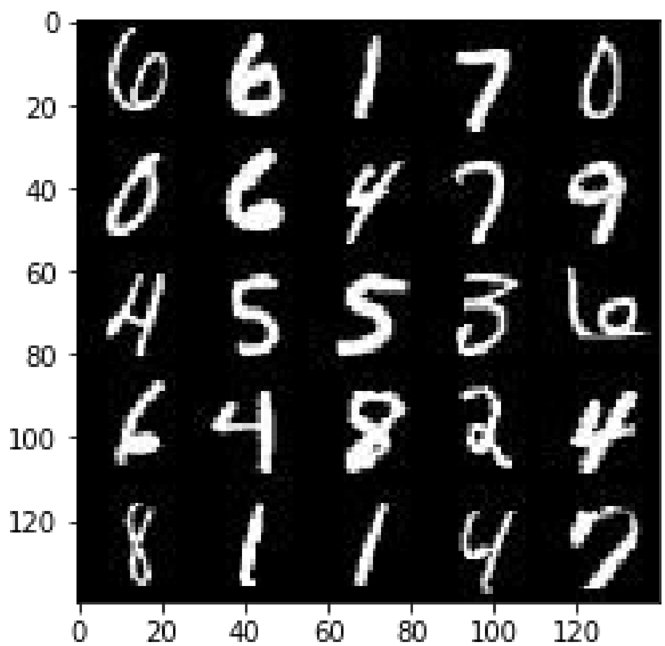
1.1 Explore the Data

1.1.1 MNIST

As you're aware, the MNIST dataset contains images of handwritten digits. You can view the first number of examples by changing `show_n_images`.

In [14]:
```
show_n_images = 25

"""
DON'T MODIFY ANYTHING IN THIS CELL
"""
%matplotlib inline
import os
from glob import glob
from matplotlib import pyplot

mnist_images = helper.get_batch(glob(os.path.join(data_dir, 'mnist/*.jpg'))[:show_n_ima
pyplot.imshow(helper.images_square_grid(mnist_images, 'L'), cmap='gray')
```

Out[14]:
```
<matplotlib.image.AxesImage at 0x7fefbe3df9b0>
```

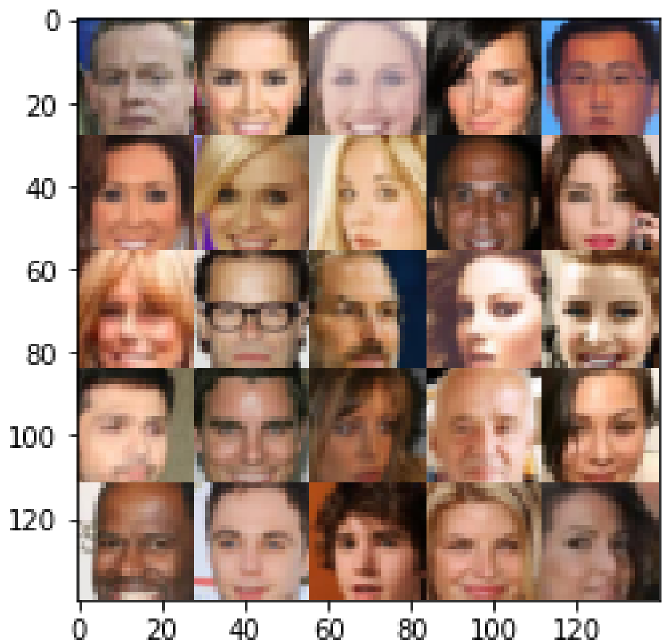


1.1.2 CelebA

The CelebFaces Attributes Dataset (CelebA) dataset contains over 200,000 celebrity images with annotations. Since you're going to be generating faces, you won't need the annotations. You can view the first number of examples by changing `show_n_images`.

In [15]:
```
show_n_images = 25

"""
DON'T MODIFY ANYTHING IN THIS CELL
"""
celeba_images = helper.get_batch(glob(os.path.join(data_dir, 'img_align_celeba/*.jpg'))
pyplot.imshow(helper.images_square_grid(celeba_images, 'RGB'))
```

Out[15]: <matplotlib.image.AxesImage at 0x7fefb64a9828>



1.2 Preprocess the Data

Since the project's main focus is on building the GANs, we'll preprocess the data for you. The values of the MNIST and CelebA dataset will be in the range of -0.5 to 0.5 of 28x28 dimensional images. The CelebA images will be cropped to remove parts of the image that don't include a face, then resized down to 28x28.

The MNIST images are black and white images with a single [color channel](https://en.wikipedia.org/wiki/Channel_(digital_image%29) while the CelebA images have [3 color channels (RGB color channel)](https://en.wikipedia.org/wiki/Channel_(digital_image%29#RGB_Images). ## Build the Neural Network You'll build the components necessary to build a GANs by implementing the following functions below: - `model_inputs` - `discriminator` - `generator` - `model_loss` - `model_opt` - `train`

1.2.1 Check the Version of TensorFlow and Access to GPU

This will check to make sure you have the correct version of TensorFlow and access to a GPU

```
In [16]: """
         DON'T MODIFY ANYTHING IN THIS CELL
         """
         from distutils.version import LooseVersion
         import warnings
         import tensorflow as tf

         # Check TensorFlow Version
         assert LooseVersion(tf.__version__) >= LooseVersion('1.0'), 'Please use TensorFlow vers
         print('TensorFlow Version: {}'.format(tf.__version__))

         # Check for a GPU
         if not tf.test.gpu_device_name():
             warnings.warn('No GPU found. Please use a GPU to train your neural network.')
         else:
             print('Default GPU Device: {}'.format(tf.test.gpu_device_name()))
```

```
TensorFlow Version: 1.3.0
Default GPU Device: /gpu:0
```

1.2.2 Input

Implement the `model_inputs` function to create TF Placeholders for the Neural Network. It should create the following placeholders: - Real input images placeholder with rank 4 using `image_width`, `image_height`, and `image_channels`. - Z input placeholder with rank 2 using `z_dim`. - Learning rate placeholder with rank 0.

Return the placeholders in the following the tuple (tensor of real input images, tensor of z data)

```
In [17]: import problem_unittests as tests

         def model_inputs(image_width, image_height, image_channels, z_dim):
             """
             Create the model inputs
             :param image_width: The input image width
             :param image_height: The input image height
             :param image_channels: The number of image channels
             :param z_dim: The dimension of Z
             :return: Tuple of (tensor of real input images, tensor of z data, learning rate)
             """
             # TODO: Implement Function
             inputs_real = tf.placeholder(tf.float32, (None, image_width, image_height, image_ch
             inputs_z = tf.placeholder(tf.float32, (None, z_dim), name='input_z')
             learning_rate = tf.placeholder(tf.float32, name = 'learning_rate')

             return inputs_real, inputs_z, learning_rate
```

```
"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
tests.test_model_inputs(model_inputs)
```

```
ERROR:tensorflow:==================================
Object was never used (type <class 'tensorflow.python.framework.ops.Operation'>):
<tf.Operation 'assert_rank_2/Assert/Assert' type=Assert>
If you want to mark it as used call its "mark_used()" method.
It was originally created here:
['File "/opt/conda/lib/python3.6/runpy.py", line 193, in _run_module_as_main\n    "__main__", mo
==================================
Tests Passed
```

1.2.3 Discriminator

Implement `discriminator` to create a discriminator neural network that discriminates on `images`. This function should be able to reuse the variables in the neural network. Use `tf.variable_scope` with a scope name of "discriminator" to allow the variables to be reused. The function should return a tuple of (tensor output of the discriminator, tensor logits of the discriminator).

```
In [18]: def leaky_relu (x,alpha=.2):
             return tf.maximum(alpha*x, x)
```

```
In [19]: def discriminator(images, reuse=False):
             """
             Create the discriminator network
             :param images: Tensor of input image(s)
             :param reuse: Boolean if the weights should be reused
             :return: Tuple of (tensor output of the discriminator, tensor logits of the discrim
             """
             with tf.variable_scope('discriminator', reuse=reuse):
                 # Input layer is 28x28x3

                 x1 = tf.layers.conv2d(images, 64, 5, strides=2, padding='same')
                 #x1 = tf.layers.batch_normalization(x1, training=True)
                 x1 = leaky_relu(x1)
                 # 14x14x64

                 x2 = tf.layers.conv2d(x1, 128, 5, strides=2, padding='same')
                 x2 = tf.layers.batch_normalization(x2, training=True)
                 x2 = leaky_relu(x2)
                 # 7x7x128

                 x3 = tf.layers.conv2d(x2, 256, 5, strides=2, padding='same')
                 x3 = tf.layers.batch_normalization(x3, training=True)
                 x3 = leaky_relu(x3)
                 # 4x4x256
```

```
        # Flatten it
        flat = tf.reshape(x3, (-1, 4*4*256))
        logits = tf.layers.dense(flat, 1)
        out = tf.sigmoid(logits)

        return out, logits


"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
tests.test_discriminator(discriminator, tf)
```

```
Tests Passed
```

1.2.4 Generator

Implement `generator` to generate an image using `z`. This function should be able to reuse the variables in the neural network. Use `tf.variable_scope` with a scope name of "generator" to allow the variables to be reused. The function should return the generated 28 x 28 x `out_channel_dim` images.

```
In [20]: def generator(z, out_channel_dim, is_train=True):
             """
             Create the generator network
             :param z: Input z
             :param out_channel_dim: The number of channels in the output image
             :param is_train: Boolean if generator is being used for training
             :return: The tensor output of the generator
             """
             # TODO: Implement Function
             with tf.variable_scope('generator', reuse= not is_train):
                 # First fully connected layer
                 x1 = tf.layers.dense(z, 7*7*512)

                 # Reshape it to start the convolutional stack
                 x1 = tf.reshape(x1, (-1, 7, 7, 512))
                 x1 = tf.layers.batch_normalization(x1, training= is_train)
                 x1 = leaky_relu(x1)
                 # 7x7x512 now

                 x2 = tf.layers.conv2d_transpose(x1, 256, 5, strides=2, padding='same')
                 x2 = tf.layers.batch_normalization(x2, training=is_train)
                 x2 = leaky_relu(x2)
                 # 14x14x256 now
```

```
        x3 = tf.layers.conv2d_transpose(x2, 64, 5, strides=2, padding='same')
        x3 = tf.layers.batch_normalization(x3, training=is_train)
        x3 = leaky_relu(x3)
        # 28x28x64 now

        x4 = tf.layers.conv2d_transpose(x3, 16, 5, strides=1, padding='same')
        x4 = tf.layers.batch_normalization(x3, training=is_train)
        x4 = leaky_relu(x3)
        # 28x28x16 now

        # Output layer
        logits = tf.layers.conv2d_transpose(x3, out_channel_dim, 3, strides=1, padding=
        # 28x28x3 now

        out = tf.tanh(logits)

        return out


"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
tests.test_generator(generator, tf)
```

```
Tests Passed
```

1.2.5 Loss

Implement `model_loss` to build the GANs for training and calculate the loss. The function should return a tuple of (discriminator loss, generator loss). Use the following functions you implemented: - `discriminator(images, reuse=False)` - `generator(z, out_channel_dim, is_train=True)`

```
In [21]: def model_loss(input_real, input_z, out_channel_dim):
             """
             Get the loss for the discriminator and generator
             :param input_real: Images from the real dataset
             :param input_z: Z input
             :param out_channel_dim: The number of channels in the output image
             :return: A tuple of (discriminator loss, generator loss)
             """
             g_model = generator(input_z, out_channel_dim)
             d_model_real, d_logits_real = discriminator(input_real)
             d_model_fake, d_logits_fake = discriminator(g_model, reuse=True)

             d_loss_real = tf.reduce_mean(
                 tf.nn.sigmoid_cross_entropy_with_logits(logits=d_logits_real, labels=tf.ones_li
```

```
    d_loss_fake = tf.reduce_mean(
        tf.nn.sigmoid_cross_entropy_with_logits(logits=d_logits_fake, labels=tf.zeros_l
    g_loss = tf.reduce_mean(
        tf.nn.sigmoid_cross_entropy_with_logits(logits=d_logits_fake, labels=tf.ones_li

    d_loss = d_loss_real + d_loss_fake

    return d_loss, g_loss


"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
tests.test_model_loss(model_loss)
```

```
Tests Passed
```

1.2.6 Optimization

Implement `model_opt` to create the optimization operations for the GANs. Use `tf.trainable_variables` to get all the trainable variables. Filter the variables with names that are in the discriminator and generator scope names. The function should return a tuple of (discriminator training operation, generator training operation).

In [22]:
```
def model_opt(d_loss, g_loss, learning_rate, beta1):
    """
    Get optimization operations
    :param d_loss: Discriminator loss Tensor
    :param g_loss: Generator loss Tensor
    :param learning_rate: Learning Rate Placeholder
    :param beta1: The exponential decay rate for the 1st moment in the optimizer
    :return: A tuple of (discriminator training operation, generator training operation
    """
    # Get weights and bias to update
    t_vars = tf.trainable_variables()
    d_vars = [var for var in t_vars if var.name.startswith('discriminator')]
    g_vars = [var for var in t_vars if var.name.startswith('generator')]

    # Optimize
    with tf.control_dependencies(tf.get_collection(tf.GraphKeys.UPDATE_OPS)):
        d_train_opt = tf.train.AdamOptimizer(learning_rate, beta1=beta1).minimize(d_los
        g_train_opt = tf.train.AdamOptimizer(learning_rate, beta1=beta1).minimize(g_los

    return d_train_opt, g_train_opt


"""
```

```
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
tests.test_model_opt(model_opt, tf)
```

```
Tests Passed
```

1.3 Neural Network Training

1.3.1 Show Output

Use this function to show the current output of the generator during training. It will help you determine how well the GANs is training.

```
In [23]: """
         DON'T MODIFY ANYTHING IN THIS CELL
         """
         import numpy as np

         def show_generator_output(sess, n_images, input_z, out_channel_dim, image_mode):
             """
             Show example output for the generator
             :param sess: TensorFlow session
             :param n_images: Number of Images to display
             :param input_z: Input Z Tensor
             :param out_channel_dim: The number of channels in the output image
             :param image_mode: The mode to use for images ("RGB" or "L")
             """
             cmap = None if image_mode == 'RGB' else 'gray'
             z_dim = input_z.get_shape().as_list()[-1]
             example_z = np.random.uniform(-1, 1, size=[n_images, z_dim])

             samples = sess.run(
                 generator(input_z, out_channel_dim, False),
                 feed_dict={input_z: example_z})

             images_grid = helper.images_square_grid(samples, image_mode)
             pyplot.imshow(images_grid, cmap=cmap)
             pyplot.show()
```

1.3.2 Train

Implement `train` to build and train the GANs. Use the following functions you implemented: - `model_inputs(image_width, image_height, image_channels, z_dim)` - `model_loss(input_real, input_z, out_channel_dim)` - `model_opt(d_loss, g_loss, learning_rate, beta1)`

Use the `show_generator_output` to show `generator` output while you train. Running `show_generator_output` for every batch will drastically increase training time and increase the size of the notebook. It's recommended to print the `generator` output every 100 batches.

In [90]:
```
def view_sample(gen_samples, data_image_mode):
    mosaic = helper.images_square_grid(gen_samples, data_image_mode)
    if data_image_mode  == 'L':
        pyplot.imshow(mosaic, cmap = 'gray')
    else:
        pyplot.imshow(mosaic)
    pyplot.show()
```

In [91]:
```
import pickle as pkl

if not os.path.exists('checkpoints'):
    os.makedirs('checkpoints')

def train(epoch_count, batch_size, z_dim, learning_rate, beta1, get_batches, data_shape
    """
    Train the GAN
    :param epoch_count: Number of epochs
    :param batch_size: Batch Size
    :param z_dim: Z dimension
    :param learning_rate: Learning Rate
    :param beta1: The exponential decay rate for the 1st moment in the optimizer
    :param get_batches: Function to get batches
    :param data_shape: Shape of the data
    :param data_image_mode: The image mode to use for images ("RGB" or "L")
    """
    print_every, show_every = 10, 100
    # build network
    input_real, input_z, LR = model_inputs(data_shape[1],data_shape[2], data_shape[3],
    d_loss, g_loss = model_loss(input_real, input_z, data_shape[-1])
    d_train_opt, g_train_opt = model_opt(d_loss, g_loss, LR, beta1)

    # init and logging
    saver = tf.train.Saver()
    sample_z = np.random.uniform(-1, 1, size=(72, z_dim))
    samples, losses = [], []
    steps = 0

    # train here
    with tf.Session() as sess:
        sess.run(tf.global_variables_initializer())
        for epoch_i in range(epoch_count):
            batch = get_batches(batch_size)
            for batch_images in batch:
                # normalized input images
                batch_images *= 2.0
                steps += 1
```

```
                # Sample random noise for G
                batch_z = np.random.uniform(-1, 1, size=(batch_size, z_dim))

                # Run optimizers
                _ = sess.run(d_train_opt, feed_dict={input_z: batch_z, input_real: batc
                _ = sess.run(g_train_opt, feed_dict={input_z: batch_z, input_real: batc

                if steps % print_every == 0:
                    # At the end of each epoch, get the losses and print them out
                    train_loss_d = d_loss.eval({input_z: batch_z, input_real: batch_ima
                    train_loss_g = g_loss.eval({input_z: batch_z, input_real: batch_ima

                    print("Epoch {}/{}...".format(epoch_i+1, epoch_count),
                          "Discriminator Loss: {:.4f}...".format(train_loss_d),
                          "Generator Loss: {:.4f}".format(train_loss_g))
                    # Save losses to view after training
                    losses.append((train_loss_d, train_loss_g))

                if steps % show_every == 0:
                    gen_samples = sess.run(generator(input_z, data_shape[-1], is_train
                                           feed_dict={input_z: sample_z})
                    samples.append(gen_samples)
                    view_sample(gen_samples,data_image_mode)

        samples.append(sess.run(generator(input_z, data_shape[-1], is_train = False), f
        saver.save(sess, './checkpoints/generator.ckpt')

    with open(data_image_mode+'_samples.pkl', 'wb') as f:
        pkl.dump(samples, f)

    return losses, samples
```

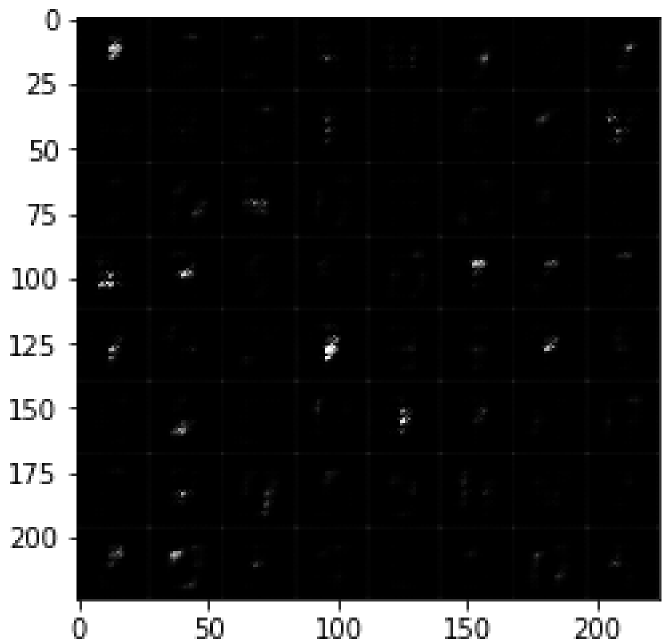
1.3.3 MNIST

Test your GANs architecture on MNIST. After 2 epochs, the GANs should be able to generate images that look like handwritten digits. Make sure the loss of the generator is lower than the loss of the discriminator or close to 0.

In [92]:
```
batch_size = 128
z_dim = 100
learning_rate = 0.0008
beta1 = 0.5


"""
```

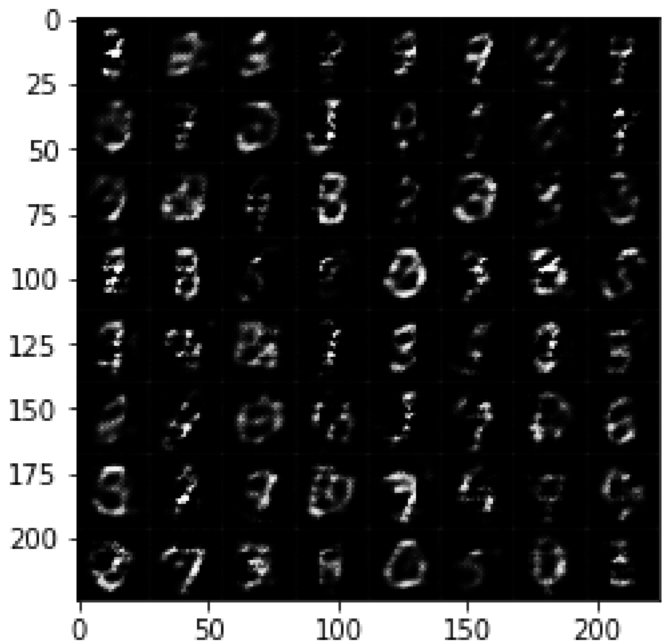
```python
    DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
    """
    epochs = 2

    mnist_dataset = helper.Dataset('mnist', glob(os.path.join(data_dir, 'mnist/*.jpg')))
    with tf.Graph().as_default():
        train(epochs, batch_size, z_dim, learning_rate, beta1, mnist_dataset.get_batches,
              mnist_dataset.shape, mnist_dataset.image_mode)
```

```
Epoch 1/2... Discriminator Loss: 1.8688... Generator Loss: 27.1767
Epoch 1/2... Discriminator Loss: 0.0050... Generator Loss: 9.9338
Epoch 1/2... Discriminator Loss: 0.0350... Generator Loss: 9.5595
Epoch 1/2... Discriminator Loss: 0.0479... Generator Loss: 5.4187
Epoch 1/2... Discriminator Loss: 0.6260... Generator Loss: 2.8535
Epoch 1/2... Discriminator Loss: 0.6554... Generator Loss: 2.5026
Epoch 1/2... Discriminator Loss: 0.5575... Generator Loss: 2.9972
Epoch 1/2... Discriminator Loss: 3.3176... Generator Loss: 3.8708
Epoch 1/2... Discriminator Loss: 0.9800... Generator Loss: 0.8161
Epoch 1/2... Discriminator Loss: 0.7419... Generator Loss: 1.7618
```

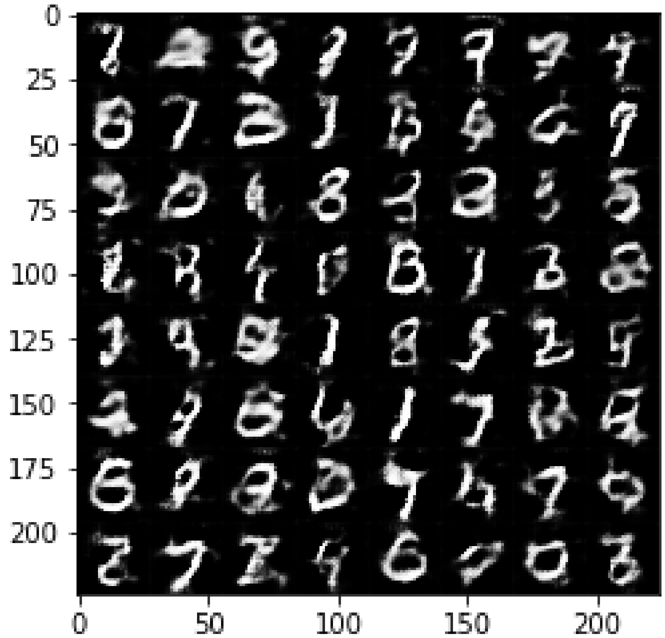



```
Epoch 1/2... Discriminator Loss: 1.1914... Generator Loss: 1.7582
Epoch 1/2... Discriminator Loss: 0.8609... Generator Loss: 1.4112
Epoch 1/2... Discriminator Loss: 1.0361... Generator Loss: 1.5803
Epoch 1/2... Discriminator Loss: 0.9709... Generator Loss: 1.2241
Epoch 1/2... Discriminator Loss: 1.4055... Generator Loss: 0.4907
```

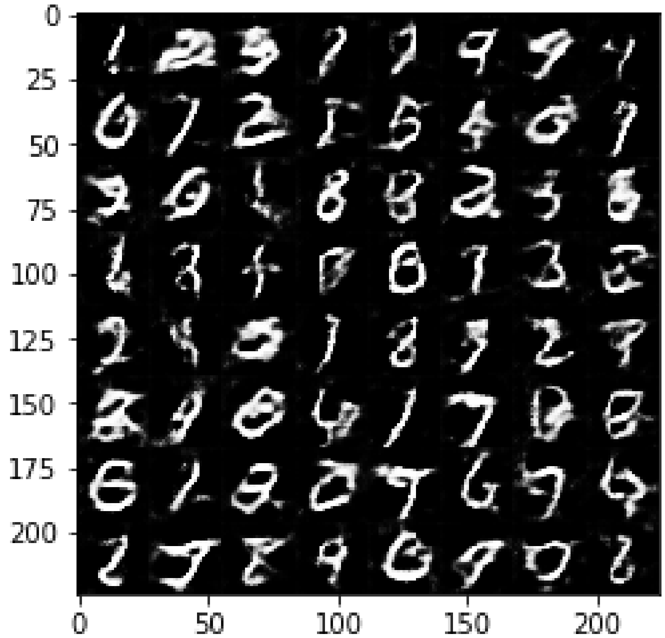
```
Epoch 1/2... Discriminator Loss: 1.3642... Generator Loss: 1.7855
Epoch 1/2... Discriminator Loss: 1.3949... Generator Loss: 1.8244
Epoch 1/2... Discriminator Loss: 1.1086... Generator Loss: 0.5706
Epoch 1/2... Discriminator Loss: 1.1134... Generator Loss: 1.3671
Epoch 1/2... Discriminator Loss: 1.0504... Generator Loss: 1.0236
```



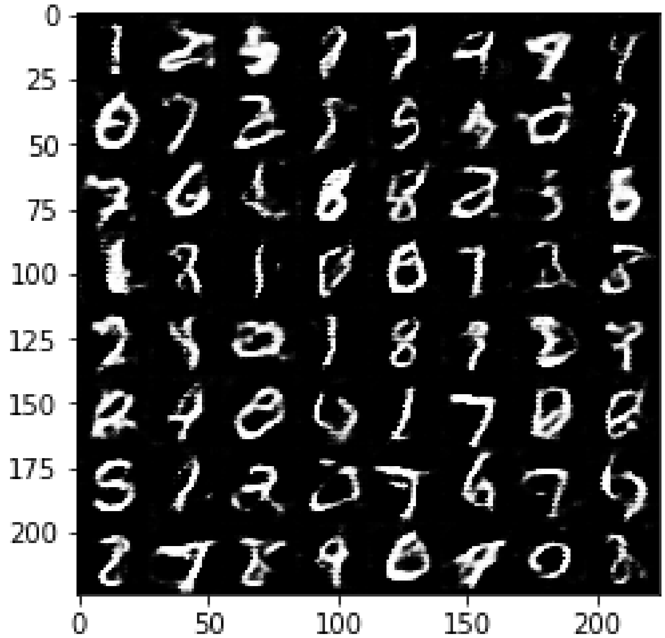

```
Epoch 1/2... Discriminator Loss: 0.9453... Generator Loss: 1.0990
Epoch 1/2... Discriminator Loss: 1.6124... Generator Loss: 0.3464
Epoch 1/2... Discriminator Loss: 0.9239... Generator Loss: 0.8711
Epoch 1/2... Discriminator Loss: 0.9714... Generator Loss: 1.1787
Epoch 1/2... Discriminator Loss: 0.7095... Generator Loss: 1.5080
Epoch 1/2... Discriminator Loss: 1.4677... Generator Loss: 0.3933
Epoch 1/2... Discriminator Loss: 1.0004... Generator Loss: 0.9370
Epoch 1/2... Discriminator Loss: 1.1647... Generator Loss: 0.6039
Epoch 1/2... Discriminator Loss: 1.2419... Generator Loss: 1.3548
Epoch 1/2... Discriminator Loss: 1.1185... Generator Loss: 1.2336
```



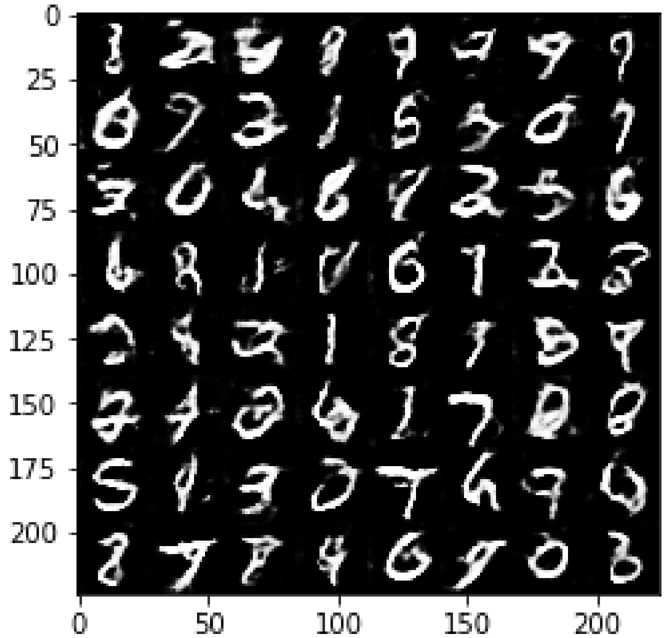

```
Epoch 1/2... Discriminator Loss: 1.1534... Generator Loss: 0.9300
Epoch 1/2... Discriminator Loss: 2.0229... Generator Loss: 2.5900
Epoch 1/2... Discriminator Loss: 1.0992... Generator Loss: 1.2135
Epoch 1/2... Discriminator Loss: 1.2337... Generator Loss: 1.1117
Epoch 1/2... Discriminator Loss: 1.2468... Generator Loss: 0.5265
Epoch 1/2... Discriminator Loss: 1.2313... Generator Loss: 0.5797
Epoch 1/2... Discriminator Loss: 1.3350... Generator Loss: 1.5621
Epoch 1/2... Discriminator Loss: 1.2554... Generator Loss: 0.6039
Epoch 1/2... Discriminator Loss: 1.1753... Generator Loss: 0.7834
Epoch 1/2... Discriminator Loss: 1.4202... Generator Loss: 0.4180
```



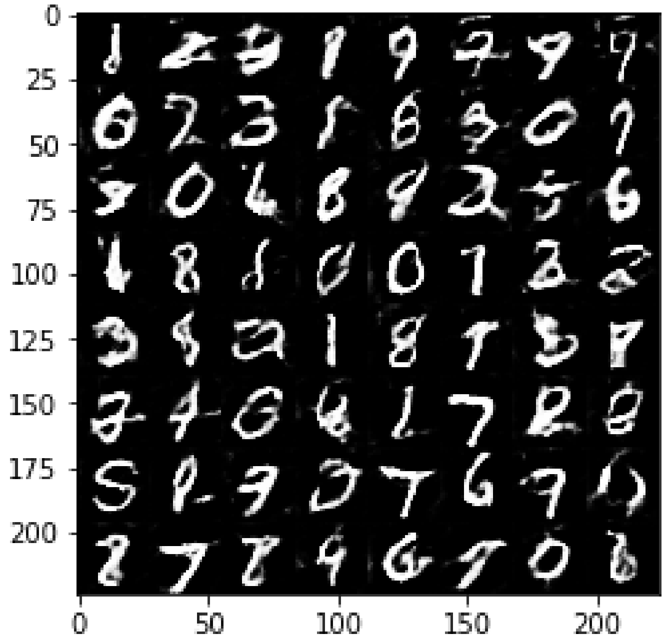

```
Epoch 1/2... Discriminator Loss: 1.2179... Generator Loss: 1.4005
Epoch 1/2... Discriminator Loss: 1.1635... Generator Loss: 1.1021
Epoch 1/2... Discriminator Loss: 1.1405... Generator Loss: 0.7712
Epoch 1/2... Discriminator Loss: 1.1528... Generator Loss: 0.7352
Epoch 1/2... Discriminator Loss: 1.1141... Generator Loss: 1.0089
Epoch 1/2... Discriminator Loss: 1.1617... Generator Loss: 0.8950
Epoch 2/2... Discriminator Loss: 1.4683... Generator Loss: 2.4230
Epoch 2/2... Discriminator Loss: 1.1650... Generator Loss: 1.2907
Epoch 2/2... Discriminator Loss: 1.1428... Generator Loss: 0.6952
Epoch 2/2... Discriminator Loss: 1.5766... Generator Loss: 2.3034
```



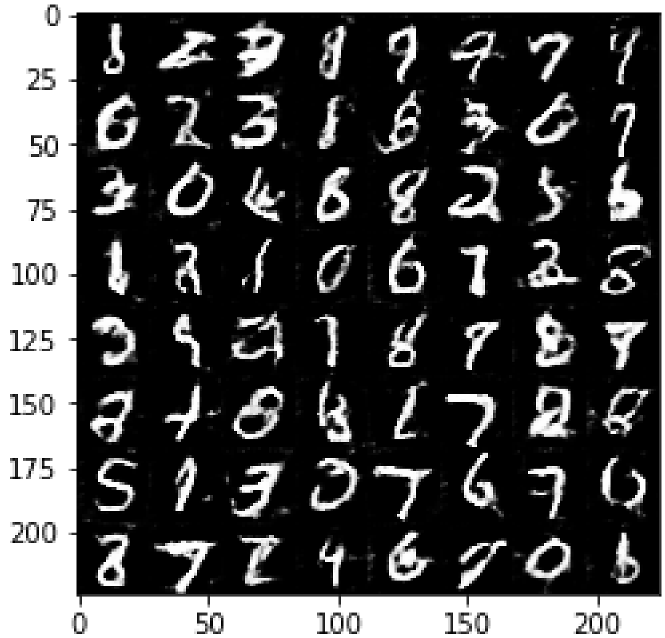

```
Epoch 2/2... Discriminator Loss: 1.1423... Generator Loss: 0.5747
Epoch 2/2... Discriminator Loss: 1.0262... Generator Loss: 0.8158
Epoch 2/2... Discriminator Loss: 1.0631... Generator Loss: 1.3159
Epoch 2/2... Discriminator Loss: 1.0809... Generator Loss: 1.0082
Epoch 2/2... Discriminator Loss: 1.0821... Generator Loss: 0.7799
Epoch 2/2... Discriminator Loss: 1.0708... Generator Loss: 1.3485
Epoch 2/2... Discriminator Loss: 1.2340... Generator Loss: 0.5954
Epoch 2/2... Discriminator Loss: 1.3772... Generator Loss: 0.4450
Epoch 2/2... Discriminator Loss: 1.2325... Generator Loss: 0.5286
Epoch 2/2... Discriminator Loss: 1.4545... Generator Loss: 0.3818
```



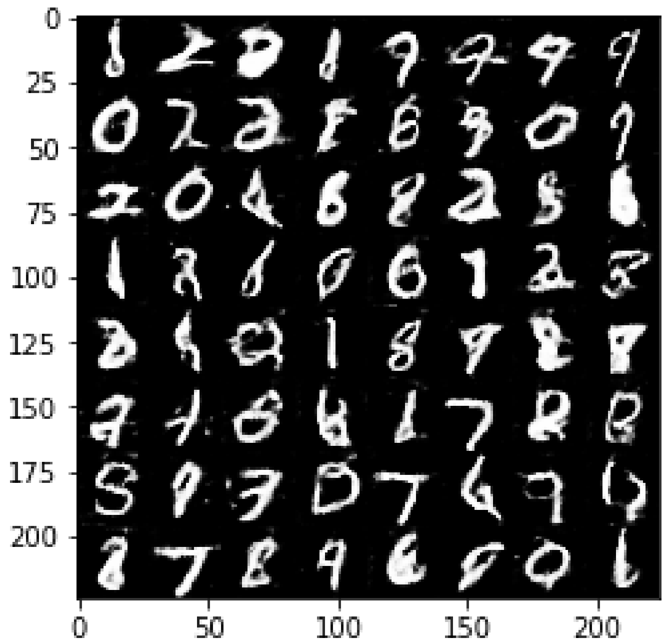

```
Epoch 2/2... Discriminator Loss: 1.1452... Generator Loss: 0.7110
Epoch 2/2... Discriminator Loss: 1.1917... Generator Loss: 0.5416
Epoch 2/2... Discriminator Loss: 1.1107... Generator Loss: 1.0463
Epoch 2/2... Discriminator Loss: 1.1145... Generator Loss: 0.7118
Epoch 2/2... Discriminator Loss: 1.0314... Generator Loss: 0.9011
Epoch 2/2... Discriminator Loss: 1.1295... Generator Loss: 0.6569
Epoch 2/2... Discriminator Loss: 1.0333... Generator Loss: 0.8514
Epoch 2/2... Discriminator Loss: 1.2479... Generator Loss: 1.1081
Epoch 2/2... Discriminator Loss: 1.2668... Generator Loss: 1.7036
Epoch 2/2... Discriminator Loss: 1.1575... Generator Loss: 0.6313
```




```
Epoch 2/2... Discriminator Loss: 1.2714... Generator Loss: 0.5228
Epoch 2/2... Discriminator Loss: 1.8242... Generator Loss: 0.2476
Epoch 2/2... Discriminator Loss: 1.0583... Generator Loss: 0.7966
Epoch 2/2... Discriminator Loss: 1.4421... Generator Loss: 0.3902
Epoch 2/2... Discriminator Loss: 1.1533... Generator Loss: 0.8628
Epoch 2/2... Discriminator Loss: 1.4031... Generator Loss: 0.3968
Epoch 2/2... Discriminator Loss: 1.2003... Generator Loss: 0.8433
Epoch 2/2... Discriminator Loss: 1.1259... Generator Loss: 0.7876
Epoch 2/2... Discriminator Loss: 1.0752... Generator Loss: 1.0215
Epoch 2/2... Discriminator Loss: 1.0805... Generator Loss: 1.3009
```

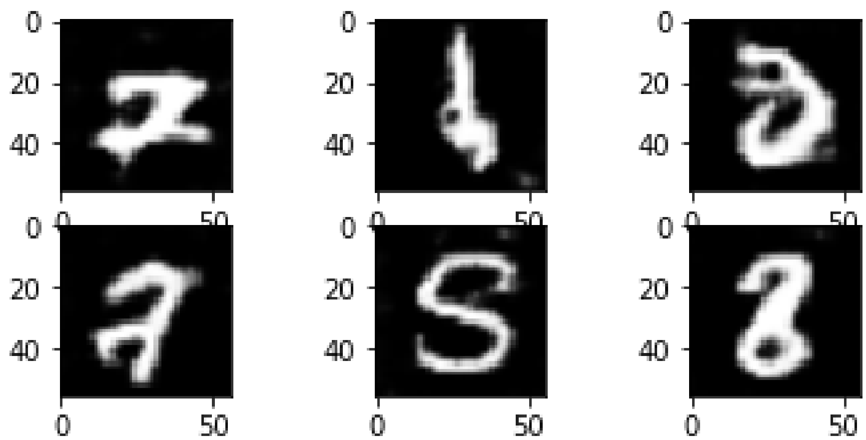



```
Epoch 2/2... Discriminator Loss: 1.0822... Generator Loss: 0.7285
Epoch 2/2... Discriminator Loss: 1.6775... Generator Loss: 0.2902
Epoch 2/2... Discriminator Loss: 1.3415... Generator Loss: 2.0439
Epoch 2/2... Discriminator Loss: 1.6224... Generator Loss: 0.3083
Epoch 2/2... Discriminator Loss: 1.0980... Generator Loss: 0.7285
Epoch 2/2... Discriminator Loss: 1.2253... Generator Loss: 0.7550
Epoch 2/2... Discriminator Loss: 1.3099... Generator Loss: 0.4579
Epoch 2/2... Discriminator Loss: 1.1302... Generator Loss: 0.9508
Epoch 2/2... Discriminator Loss: 1.1158... Generator Loss: 1.6425
Epoch 2/2... Discriminator Loss: 1.1679... Generator Loss: 1.2964
```




```
Epoch 2/2... Discriminator Loss: 1.0827... Generator Loss: 0.7405
Epoch 2/2... Discriminator Loss: 1.0123... Generator Loss: 0.8565
Epoch 2/2... Discriminator Loss: 1.2286... Generator Loss: 0.6595
```

In [165]:
```python
from scipy.misc import imresize
hw_samples = pkl.load(open('L_samples.pkl', 'rb'))
for i in range(6):
    pyplot.subplot(3,3,1+i)
    pyplot.imshow(imresize(hw_samples[-1][i+2,:,:,0], [56,56]), cmap = 'gray')
```

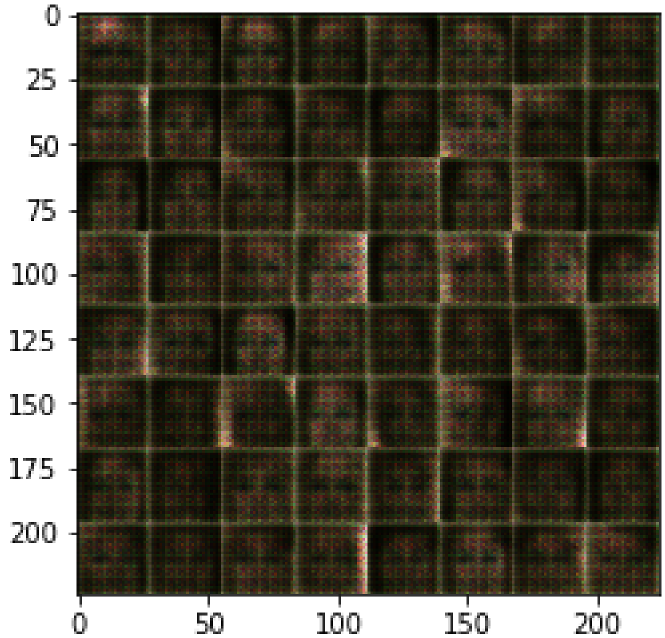


1.3.4 CelebA

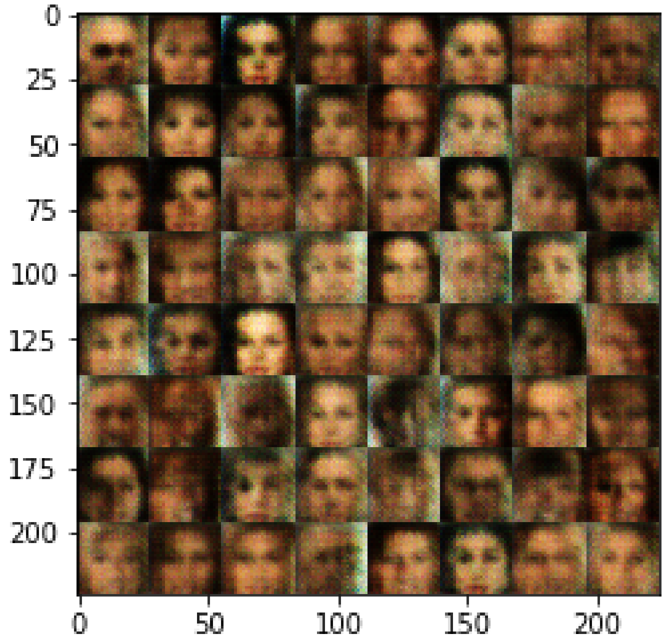
Run your GANs on CelebA. It will take around 20 minutes on the average GPU to run one epoch. You can run the whole epoch or stop when it starts to generate realistic faces.

In [166]:
```
batch_size = 512
z_dim = 200
learning_rate = 0.0002
beta1 = 0.5

"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
epochs = 1

celeba_dataset = helper.Dataset('celeba', glob(os.path.join(data_dir, 'img_align_celeb
with tf.Graph().as_default():
    train(epochs, batch_size, z_dim, learning_rate, beta1, celeba_dataset.get_batches,
          celeba_dataset.shape, celeba_dataset.image_mode)
```

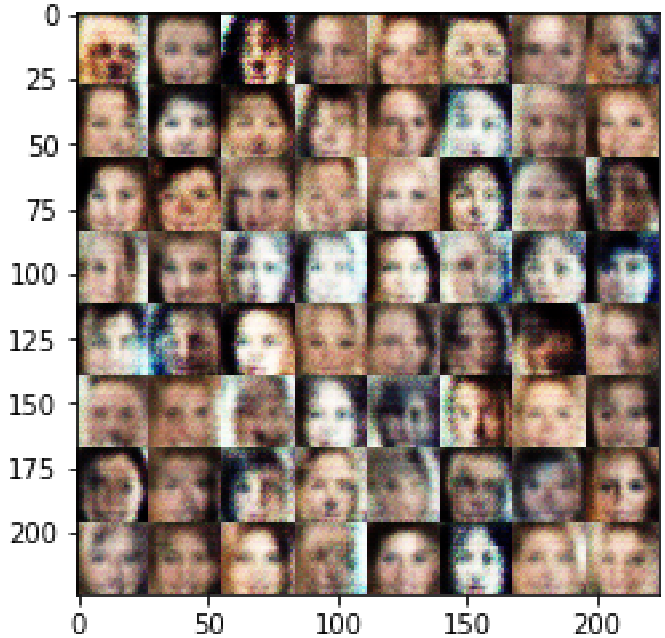
```
Epoch 1/1... Discriminator Loss: 0.3182... Generator Loss: 2.5014
Epoch 1/1... Discriminator Loss: 0.2743... Generator Loss: 2.3291
Epoch 1/1... Discriminator Loss: 0.7220... Generator Loss: 0.9050
Epoch 1/1... Discriminator Loss: 0.2081... Generator Loss: 2.5578
Epoch 1/1... Discriminator Loss: 0.4899... Generator Loss: 4.4986
Epoch 1/1... Discriminator Loss: 1.9820... Generator Loss: 3.8334
Epoch 1/1... Discriminator Loss: 0.8845... Generator Loss: 0.9188
Epoch 1/1... Discriminator Loss: 0.7283... Generator Loss: 2.2918
Epoch 1/1... Discriminator Loss: 1.0728... Generator Loss: 1.0399
Epoch 1/1... Discriminator Loss: 1.0309... Generator Loss: 2.0679
```




```
Epoch 1/1... Discriminator Loss: 0.7409... Generator Loss: 1.1180
Epoch 1/1... Discriminator Loss: 0.8910... Generator Loss: 2.4534
Epoch 1/1... Discriminator Loss: 0.7630... Generator Loss: 1.5896
Epoch 1/1... Discriminator Loss: 1.5993... Generator Loss: 3.3810
Epoch 1/1... Discriminator Loss: 2.1551... Generator Loss: 0.2166
Epoch 1/1... Discriminator Loss: 2.8125... Generator Loss: 0.1085
Epoch 1/1... Discriminator Loss: 1.8295... Generator Loss: 2.5180
Epoch 1/1... Discriminator Loss: 0.7571... Generator Loss: 1.5443
Epoch 1/1... Discriminator Loss: 0.7799... Generator Loss: 2.5464
Epoch 1/1... Discriminator Loss: 0.5139... Generator Loss: 2.2083
```

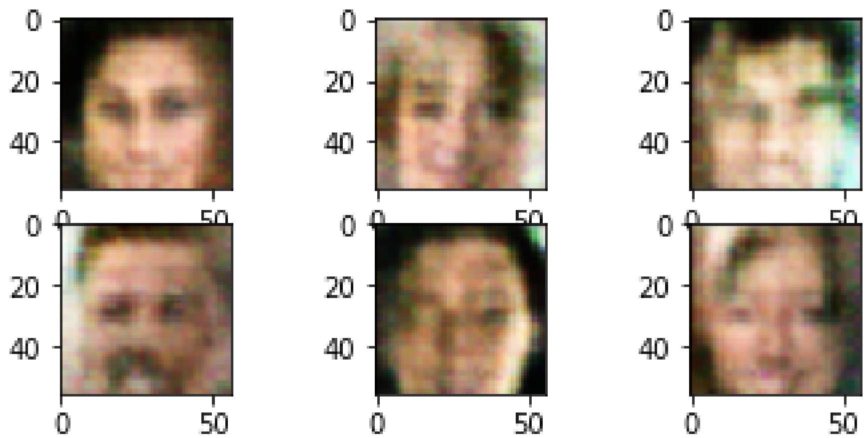



```
Epoch 1/1... Discriminator Loss: 0.9113... Generator Loss: 0.8548
Epoch 1/1... Discriminator Loss: 0.9338... Generator Loss: 0.7951
Epoch 1/1... Discriminator Loss: 0.8679... Generator Loss: 0.9326
Epoch 1/1... Discriminator Loss: 2.3947... Generator Loss: 0.1413
Epoch 1/1... Discriminator Loss: 0.9904... Generator Loss: 0.9101
Epoch 1/1... Discriminator Loss: 1.4871... Generator Loss: 2.3024
Epoch 1/1... Discriminator Loss: 1.2810... Generator Loss: 0.8322
Epoch 1/1... Discriminator Loss: 1.3820... Generator Loss: 1.8376
Epoch 1/1... Discriminator Loss: 1.6478... Generator Loss: 0.7311
Epoch 1/1... Discriminator Loss: 1.4036... Generator Loss: 0.7845
```




```
Epoch 1/1... Discriminator Loss: 1.1022... Generator Loss: 1.2983
Epoch 1/1... Discriminator Loss: 1.3266... Generator Loss: 0.6967
Epoch 1/1... Discriminator Loss: 1.3437... Generator Loss: 0.7558
Epoch 1/1... Discriminator Loss: 1.4833... Generator Loss: 0.6419
Epoch 1/1... Discriminator Loss: 1.4533... Generator Loss: 0.7054
Epoch 1/1... Discriminator Loss: 1.4912... Generator Loss: 0.7257
Epoch 1/1... Discriminator Loss: 1.3694... Generator Loss: 0.6991
Epoch 1/1... Discriminator Loss: 1.4286... Generator Loss: 0.7215
Epoch 1/1... Discriminator Loss: 1.3672... Generator Loss: 0.7761
```

In [167]:
```
face_samples = pkl.load(open('RGB_samples.pkl', 'rb'))
for i in range(6):
    pyplot.subplot(3,3,1+i)
    pyplot.imshow(imresize(face_samples[-1][i+2,:,:,:], [56,56,3]))
```



1.3.5 Submitting This Project

When submitting this project, make sure to run all the cells before saving the notebook. Save the notebook file as "dlnd_face_generation.ipynb" and save it as a HTML file under "File" -> "Download as". Include the "helper.py" and "problem_unittests.py" files in your submission.