GAN_mnist_v2

January 11, 2019

```
In [43]: from __future__ import absolute_import
         from __future__ import division
         from __future__ import print_function

         from keras.layers import Activation, Dense, Input
         from keras.layers import Conv2D, Flatten
         from keras.layers import Reshape, Conv2DTranspose
         from keras.layers import LeakyReLU
         from keras.layers import BatchNormalization
         from keras import optimizers
         from keras.models import Model
         from keras.datasets import mnist
         from keras.models import load_model

In [44]: import numpy as np
         import math
         import os
         import matplotlib.pyplot as plt

In [45]: def NN_discriminator(inputs,kernel_size):
             #kernel_size: length of CNN window
             #input: image
             nfilter_layers=[16,32,64,128]
             x=inputs
             for nfilter in nfilter_layers:
                 stride=2
                 if nfilter==nfilter_layers[-1]:
                     stride=1
                 x=LeakyReLU(alpha=0.2)(x)
                 x=Conv2D(filters=nfilter,kernel_size=kernel_size,strides=stride,padding='same')
             x=Flatten()(x)
             x=Dense(1)(x)
             x=Activation('sigmoid')(x)
             discriminator=Model(inputs,x,name='Discriminator')
             return discriminator

In [46]: def NN_generator(inputs,size_imag,kernel_size):
             nfilter_layers=[64,32,16,1]
```

```
    imag_resize=size_imag//4
    x=Dense(imag_resize*imag_resize*nfilter_layers[0])(inputs)
    x=Reshape((imag_resize,imag_resize,nfilter_layers[0]))(x)

    for nfilter in nfilter_layers:
        stride=2
        if nfilter>nfilter_layers[-2]:
            stride=1
        x=BatchNormalization()(x)
        x=Activation('relu')(x)
        x=Conv2DTranspose(filters=nfilter,kernel_size=kernel_size,strides=stride,paddin
    x=Activation('sigmoid')(x)
    generator=Model(inputs,x,name='Generator')
    return generator
```

```
In [47]: def plot_imag(generator,noise_input,show=False,step=0,model_name='gan'):
             """
             generator(model)
             noise_input(ndarray): array of latent vector
             """
             os.makedirs(model_name,exist_ok=True)
             filename=os.path.join(model_name,"%3d.png" %step)
             imags=generator.predict(noise_input)
             plt.figure(figsize=(10,10))
             n_imag=imags.shape[0]
             size_imag=imags.shape[1]
             cols=4
             rows=int(noise_input.shape[0]/cols)
             for i in range(n_imag):
                 plt.subplot(rows,cols,i+1)
                 imag=np.reshape(imags[i],[size_imag,size_imag])
                 plt.imshow(imag,cmap='gray')
                 plt.axis('off')
             plt.savefig(filename)
             if show:
                 plt.show()
             else:
                 plt.close('all')
```

```
In [48]: def train_GAN(models,x_train,param):
             generator,discrim,gan=models
             batch_size, dim_latent, train_steps = param
             save_interval=1000
             noise=np.random.uniform(-1.0,1.0,size=[16,dim_latent])
             train_size=x_train.shape[0]
             for i in range(train_steps):
                 id_rand=np.random.randint(0,train_size,size=batch_size)
```

```
            imag_real=x_train[id_rand]
            noise_input=np.random.uniform(-1.0,1.0,size=[batch_size,dim_latent])
            imag_fake=generator.predict(noise_input)
            x=np.concatenate((imag_real,imag_fake))
            y=np.ones([2*batch_size,1])
            y[batch_size:,:]=0.0
            loss,acc=discrim.train_on_batch(x,y)
            if i>=100 and i%100==0:
                log="%d:discrim loss: %f, acc: %f" %(i,loss,acc)

            noise=np.random.uniform(-1.0,1.0,size=[batch_size,dim_latent])
            y=np.ones([batch_size,1])
            loss,acc=gan.train_on_batch(noise,y)
            if i>=100 and i%100==0:
                log="%s; gan loss: %f, acc: %f" %(log,loss,acc)
                print(log)
            if(i+1)% save_interval==0:
                if (i+1)==train_steps:
                    show_flag=True
                else:
                    show_flag=False
                plot_imag(generator,noise_input=noise_input,show=show_flag,step=(i+1),model
        generator.save("gan_mnist.h5")
```

```
In [49]: def build_GAN():
             (x_train,_),(_,_)=mnist.load_data()
             size_imag=x_train.shape[1]
             x_train=np.reshape(x_train,[-1,size_imag,size_imag,1])
             x_train=x_train.astype('float32')/255

             dim_latent=25
             batch_size=64
             train_steps=10000
             lr=2e-4
             decay_rate = 1e-7

             #build discriminator model
             input_shape=(size_imag,size_imag,1)
             input_discrim=Input(shape=input_shape,name='input_discrim')
             kernel_size_discrim=5
             discrim=NN_discriminator(input_discrim,kernel_size_discrim)
             optimizer=optimizers.Adam(lr=lr,decay=decay_rate)
             discrim.compile(loss='binary_crossentropy',optimizer=optimizer,metrics=['accuracy']
             discrim.summary()

             #build generator model
             kernel_size_gen=5
```

```
        input_shape=(dim_latent,)
        input_gen=Input(shape=input_shape,name='latent_input')
        generator=NN_generator(input_gen,size_imag,kernel_size_gen)
        #generator.compile(loss='binary_crossentropy',optimizer=optimizer,metrics=['accurac
        generator.summary()

        optimizer=optimizers.Adam(lr=lr,decay=decay_rate*0.5)
        discrim.trainable = False
        gan=Model(input_gen,discrim(generator(input_gen)),name='gan')
        gan.compile(loss='binary_crossentropy',optimizer=optimizer,metrics=['accuracy'])
        gan.summary()

        models=(generator,discrim,gan)
        param=(batch_size,dim_latent,train_steps)
        train_GAN(models,x_train,param)
```

```
In [50]: def test_generator(generator):
             noise_input=np.random.uniform(-1.0,1.0,size=[16,20])
             plot_imag(generator,noise_input,show=True,model_name='test_generator')
```

```
In [51]: build_GAN()
```

```
WARNING:tensorflow:From /Users/Bluesky/anaconda/lib/python3.6/site-packages/keras/backend/tensor
Instructions for updating:
keep_dims is deprecated, use keepdims instead
WARNING:tensorflow:From /Users/Bluesky/anaconda/lib/python3.6/site-packages/keras/backend/tensor
Instructions for updating:
keep_dims is deprecated, use keepdims instead
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_discrim (InputLayer)   (None, 28, 28, 1)         0         
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 28, 28, 1)         0         
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 14, 14, 16)        416       
_________________________________________________________________
leaky_re_lu_2 (LeakyReLU)    (None, 14, 14, 16)        0         
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 7, 7, 32)          12832     
_________________________________________________________________
leaky_re_lu_3 (LeakyReLU)    (None, 7, 7, 32)          0         
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 4, 4, 64)          51264     
_________________________________________________________________
leaky_re_lu_4 (LeakyReLU)    (None, 4, 4, 64)          0         
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 4, 4, 128)         204928    
```

```
_________________________________________________________________
flatten_1 (Flatten)          (None, 2048)              0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 2049
_________________________________________________________________
activation_1 (Activation)    (None, 1)                 0
=================================================================
Total params: 271,489
Trainable params: 271,489
Non-trainable params: 0
_________________________________________________________________
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
latent_input (InputLayer)    (None, 25)                0
_________________________________________________________________
dense_2 (Dense)              (None, 3136)              81536
_________________________________________________________________
reshape_1 (Reshape)          (None, 7, 7, 64)          0
_________________________________________________________________
batch_normalization_1 (Batch (None, 7, 7, 64)          256
_________________________________________________________________
activation_2 (Activation)    (None, 7, 7, 64)          0
_________________________________________________________________
conv2d_transpose_1 (Conv2DTr (None, 7, 7, 64)          102464
_________________________________________________________________
batch_normalization_2 (Batch (None, 7, 7, 64)          256
_________________________________________________________________
activation_3 (Activation)    (None, 7, 7, 64)          0
_________________________________________________________________
conv2d_transpose_2 (Conv2DTr (None, 7, 7, 32)          51232
_________________________________________________________________
batch_normalization_3 (Batch (None, 7, 7, 32)          128
_________________________________________________________________
activation_4 (Activation)    (None, 7, 7, 32)          0
_________________________________________________________________
conv2d_transpose_3 (Conv2DTr (None, 14, 14, 16)        12816
_________________________________________________________________
batch_normalization_4 (Batch (None, 14, 14, 16)        64
_________________________________________________________________
activation_5 (Activation)    (None, 14, 14, 16)        0
_________________________________________________________________
conv2d_transpose_4 (Conv2DTr (None, 28, 28, 1)         401
_________________________________________________________________
activation_6 (Activation)    (None, 28, 28, 1)         0
=================================================================
Total params: 249,153
Trainable params: 248,801
```

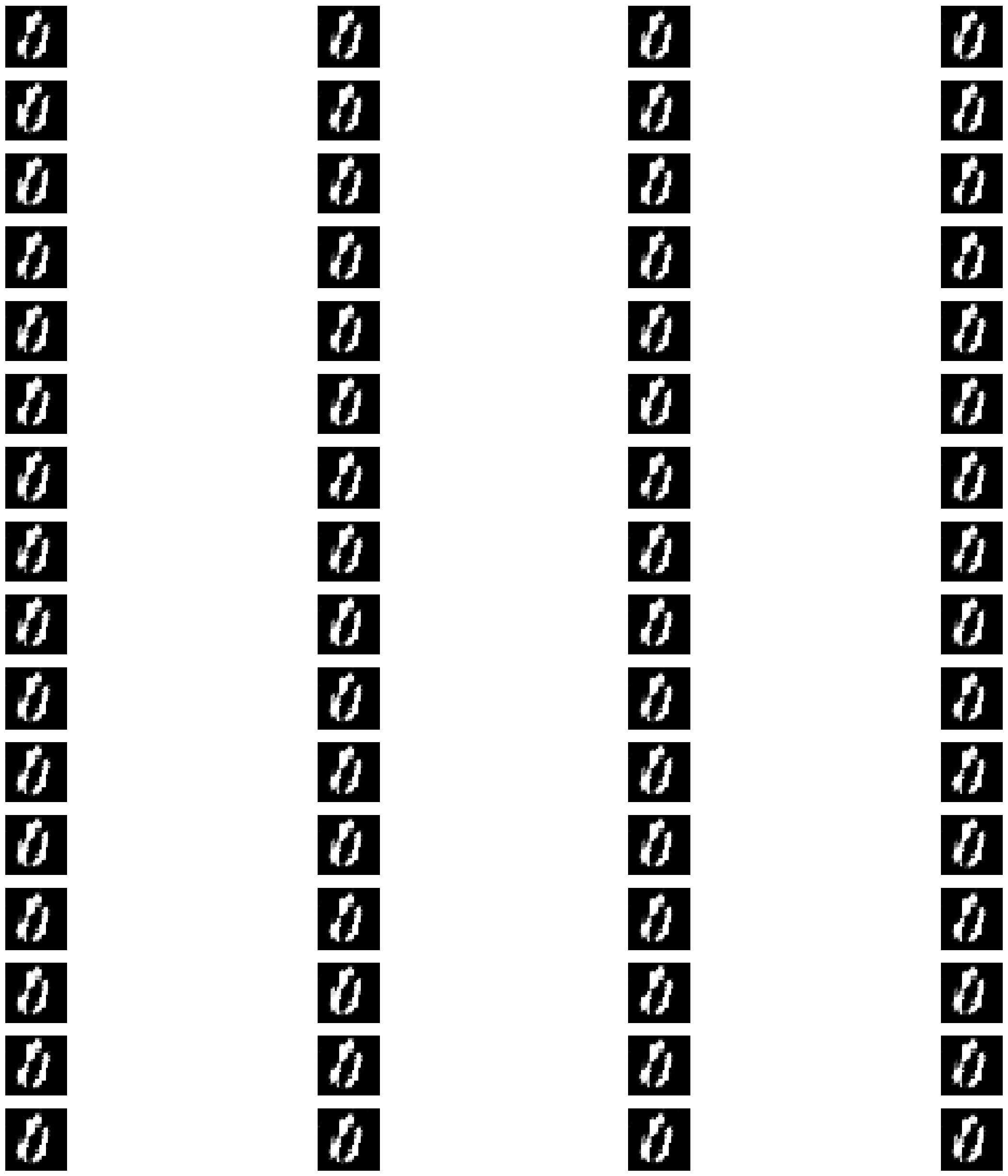
```
Non-trainable params: 352
_________________________________________________________________
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
latent_input (InputLayer)    (None, 25)                0
_________________________________________________________________
Generator (Model)            (None, 28, 28, 1)         249153
_________________________________________________________________
Discriminator (Model)        (None, 1)                 271489
=================================================================
Total params: 520,642
Trainable params: 248,801
Non-trainable params: 271,841
_________________________________________________________________


/Users/Bluesky/anaconda/lib/python3.6/site-packages/keras/engine/training.py:973: UserWarning: D
  'Discrepancy between trainable weights and collected trainable'


WARNING:tensorflow:Variable *= will be deprecated. Use `var.assign(var * other)` if you want ass
100:discrim loss: 0.000439, acc: 1.000000; gan loss: 0.100005, acc: 1.000000
200:discrim loss: 0.001729, acc: 1.000000; gan loss: 0.696779, acc: 0.375000
300:discrim loss: 0.014209, acc: 1.000000; gan loss: 3.092068, acc: 0.000000
400:discrim loss: 0.007476, acc: 1.000000; gan loss: 7.595908, acc: 0.000000
500:discrim loss: 0.005103, acc: 1.000000; gan loss: 10.841255, acc: 0.000000
600:discrim loss: 0.010750, acc: 0.992188; gan loss: 10.799322, acc: 0.000000
700:discrim loss: 0.016669, acc: 0.992188; gan loss: 7.734886, acc: 0.000000
800:discrim loss: 0.066661, acc: 0.984375; gan loss: 0.278635, acc: 0.921875
900:discrim loss: 0.033316, acc: 0.984375; gan loss: 2.809544, acc: 0.000000
1000:discrim loss: 0.096303, acc: 0.984375; gan loss: 6.335379, acc: 0.000000
1100:discrim loss: 0.135610, acc: 0.968750; gan loss: 7.742282, acc: 0.000000
1200:discrim loss: 0.056056, acc: 0.992188; gan loss: 6.527595, acc: 0.000000
1300:discrim loss: 0.007882, acc: 1.000000; gan loss: 6.837227, acc: 0.000000
1400:discrim loss: 0.000948, acc: 1.000000; gan loss: 5.883149, acc: 0.000000
1500:discrim loss: 0.110971, acc: 0.984375; gan loss: 2.406357, acc: 0.000000
1600:discrim loss: 0.032289, acc: 0.976562; gan loss: 0.440018, acc: 0.937500
1700:discrim loss: 0.011915, acc: 1.000000; gan loss: 0.723585, acc: 0.500000
1800:discrim loss: 0.070145, acc: 0.984375; gan loss: 6.081949, acc: 0.000000
1900:discrim loss: 0.022849, acc: 1.000000; gan loss: 2.097632, acc: 0.000000
2000:discrim loss: 0.019496, acc: 0.984375; gan loss: 5.427408, acc: 0.000000
2100:discrim loss: 0.010410, acc: 1.000000; gan loss: 3.820832, acc: 0.000000
2200:discrim loss: 0.004817, acc: 1.000000; gan loss: 0.577013, acc: 0.843750
2300:discrim loss: 0.401362, acc: 0.929688; gan loss: 1.619822, acc: 0.031250
2400:discrim loss: 0.035115, acc: 1.000000; gan loss: 7.548785, acc: 0.000000
2500:discrim loss: 0.302742, acc: 0.937500; gan loss: 4.510469, acc: 0.000000
2600:discrim loss: 0.093801, acc: 0.992188; gan loss: 3.382614, acc: 0.000000
```

```
2700:discrim loss: 0.190721, acc: 0.953125; gan loss: 2.945575, acc: 0.000000
2800:discrim loss: 0.563952, acc: 0.890625; gan loss: 1.653287, acc: 0.093750
2900:discrim loss: 0.051340, acc: 0.984375; gan loss: 3.203661, acc: 0.000000
3000:discrim loss: 0.061976, acc: 0.984375; gan loss: 3.761132, acc: 0.000000
3100:discrim loss: 0.053530, acc: 0.976562; gan loss: 8.308662, acc: 0.000000
3200:discrim loss: 0.058595, acc: 0.992188; gan loss: 4.958867, acc: 0.000000
3300:discrim loss: 0.121906, acc: 0.960938; gan loss: 3.018098, acc: 0.000000
3400:discrim loss: 0.246590, acc: 0.945312; gan loss: 3.430802, acc: 0.000000
3500:discrim loss: 0.011850, acc: 1.000000; gan loss: 3.231153, acc: 0.000000
3600:discrim loss: 0.012579, acc: 1.000000; gan loss: 4.086392, acc: 0.000000
3700:discrim loss: 0.170534, acc: 0.960938; gan loss: 3.528139, acc: 0.000000
3800:discrim loss: 0.027985, acc: 0.992188; gan loss: 0.383569, acc: 0.984375
3900:discrim loss: 0.041511, acc: 1.000000; gan loss: 4.723305, acc: 0.000000
4000:discrim loss: 0.002422, acc: 1.000000; gan loss: 5.421032, acc: 0.000000
4100:discrim loss: 0.436738, acc: 0.929688; gan loss: 10.192231, acc: 0.000000
4200:discrim loss: 0.055449, acc: 0.984375; gan loss: 5.755372, acc: 0.000000
4300:discrim loss: 0.010675, acc: 1.000000; gan loss: 6.697473, acc: 0.000000
4400:discrim loss: 0.053408, acc: 1.000000; gan loss: 4.973292, acc: 0.000000
4500:discrim loss: 0.001720, acc: 1.000000; gan loss: 5.764590, acc: 0.000000
4600:discrim loss: 0.075048, acc: 0.984375; gan loss: 4.910754, acc: 0.000000
4700:discrim loss: 0.005435, acc: 1.000000; gan loss: 4.153384, acc: 0.000000
4800:discrim loss: 0.006490, acc: 1.000000; gan loss: 4.917475, acc: 0.000000
4900:discrim loss: 0.028167, acc: 0.992188; gan loss: 4.515633, acc: 0.000000
5000:discrim loss: 0.008199, acc: 1.000000; gan loss: 3.983949, acc: 0.000000
5100:discrim loss: 0.321443, acc: 0.953125; gan loss: 10.567509, acc: 0.000000
5200:discrim loss: 0.009245, acc: 1.000000; gan loss: 8.321796, acc: 0.000000
5300:discrim loss: 0.002848, acc: 1.000000; gan loss: 1.634479, acc: 0.000000
5400:discrim loss: 0.009158, acc: 0.992188; gan loss: 7.042754, acc: 0.000000
5500:discrim loss: 0.037139, acc: 0.992188; gan loss: 2.684925, acc: 0.000000
5600:discrim loss: 0.153389, acc: 0.984375; gan loss: 5.108039, acc: 0.000000
5700:discrim loss: 0.008777, acc: 0.992188; gan loss: 3.501237, acc: 0.000000
5800:discrim loss: 0.066438, acc: 0.984375; gan loss: 6.923916, acc: 0.000000
5900:discrim loss: 0.003124, acc: 1.000000; gan loss: 5.505821, acc: 0.000000
6000:discrim loss: 0.004663, acc: 1.000000; gan loss: 13.488536, acc: 0.000000
6100:discrim loss: 0.005920, acc: 1.000000; gan loss: 3.868663, acc: 0.000000
6200:discrim loss: 0.207454, acc: 0.921875; gan loss: 4.320680, acc: 0.000000
6300:discrim loss: 0.002648, acc: 1.000000; gan loss: 6.466701, acc: 0.000000
6400:discrim loss: 0.014372, acc: 0.992188; gan loss: 10.014089, acc: 0.000000
6500:discrim loss: 0.238421, acc: 0.976562; gan loss: 3.022755, acc: 0.046875
6600:discrim loss: 0.070935, acc: 0.984375; gan loss: 11.848810, acc: 0.000000
6700:discrim loss: 0.002549, acc: 1.000000; gan loss: 1.725040, acc: 0.000000
6800:discrim loss: 0.002574, acc: 1.000000; gan loss: 7.561848, acc: 0.000000
6900:discrim loss: 0.004695, acc: 1.000000; gan loss: 3.688414, acc: 0.000000
7000:discrim loss: 0.002477, acc: 1.000000; gan loss: 6.038299, acc: 0.000000
7100:discrim loss: 0.006429, acc: 1.000000; gan loss: 7.639956, acc: 0.000000
7200:discrim loss: 0.006732, acc: 1.000000; gan loss: 3.496968, acc: 0.000000
7300:discrim loss: 0.013261, acc: 1.000000; gan loss: 6.062716, acc: 0.000000
7400:discrim loss: 0.026811, acc: 1.000000; gan loss: 4.834062, acc: 0.000000
```

```
7500:discrim loss: 0.057601, acc: 1.000000; gan loss: 4.519482, acc: 0.000000
7600:discrim loss: 0.018684, acc: 0.992188; gan loss: 7.055186, acc: 0.000000
7700:discrim loss: 0.068512, acc: 0.976562; gan loss: 3.647113, acc: 0.015625
7800:discrim loss: 0.004617, acc: 1.000000; gan loss: 5.494446, acc: 0.000000
7900:discrim loss: 0.000499, acc: 1.000000; gan loss: 6.044356, acc: 0.000000
8000:discrim loss: 0.001179, acc: 1.000000; gan loss: 6.303755, acc: 0.000000
8100:discrim loss: 0.006440, acc: 1.000000; gan loss: 7.275563, acc: 0.000000
8200:discrim loss: 0.008912, acc: 1.000000; gan loss: 5.362094, acc: 0.000000
8300:discrim loss: 0.008547, acc: 1.000000; gan loss: 4.968718, acc: 0.000000
8400:discrim loss: 0.021923, acc: 0.992188; gan loss: 4.417671, acc: 0.000000
8500:discrim loss: 0.067454, acc: 0.992188; gan loss: 4.163150, acc: 0.000000
8600:discrim loss: 0.003095, acc: 1.000000; gan loss: 5.287883, acc: 0.000000
8700:discrim loss: 0.027734, acc: 0.992188; gan loss: 3.938588, acc: 0.000000
8800:discrim loss: 0.010159, acc: 1.000000; gan loss: 5.268146, acc: 0.000000
8900:discrim loss: 0.007085, acc: 1.000000; gan loss: 5.170542, acc: 0.000000
9000:discrim loss: 0.000304, acc: 1.000000; gan loss: 9.278217, acc: 0.000000
9100:discrim loss: 0.002476, acc: 1.000000; gan loss: 5.768137, acc: 0.000000
9200:discrim loss: 0.012608, acc: 1.000000; gan loss: 4.816066, acc: 0.000000
9300:discrim loss: 0.025329, acc: 0.992188; gan loss: 6.931050, acc: 0.000000
9400:discrim loss: 0.017296, acc: 0.992188; gan loss: 5.610292, acc: 0.000000
9500:discrim loss: 0.000680, acc: 1.000000; gan loss: 6.575785, acc: 0.000000
9600:discrim loss: 0.000274, acc: 1.000000; gan loss: 7.872110, acc: 0.000000
9700:discrim loss: 0.001875, acc: 1.000000; gan loss: 6.029726, acc: 0.000000
9800:discrim loss: 0.054748, acc: 0.984375; gan loss: 7.870944, acc: 0.000000
9900:discrim loss: 0.092856, acc: 0.984375; gan loss: 9.778940, acc: 0.000000
```

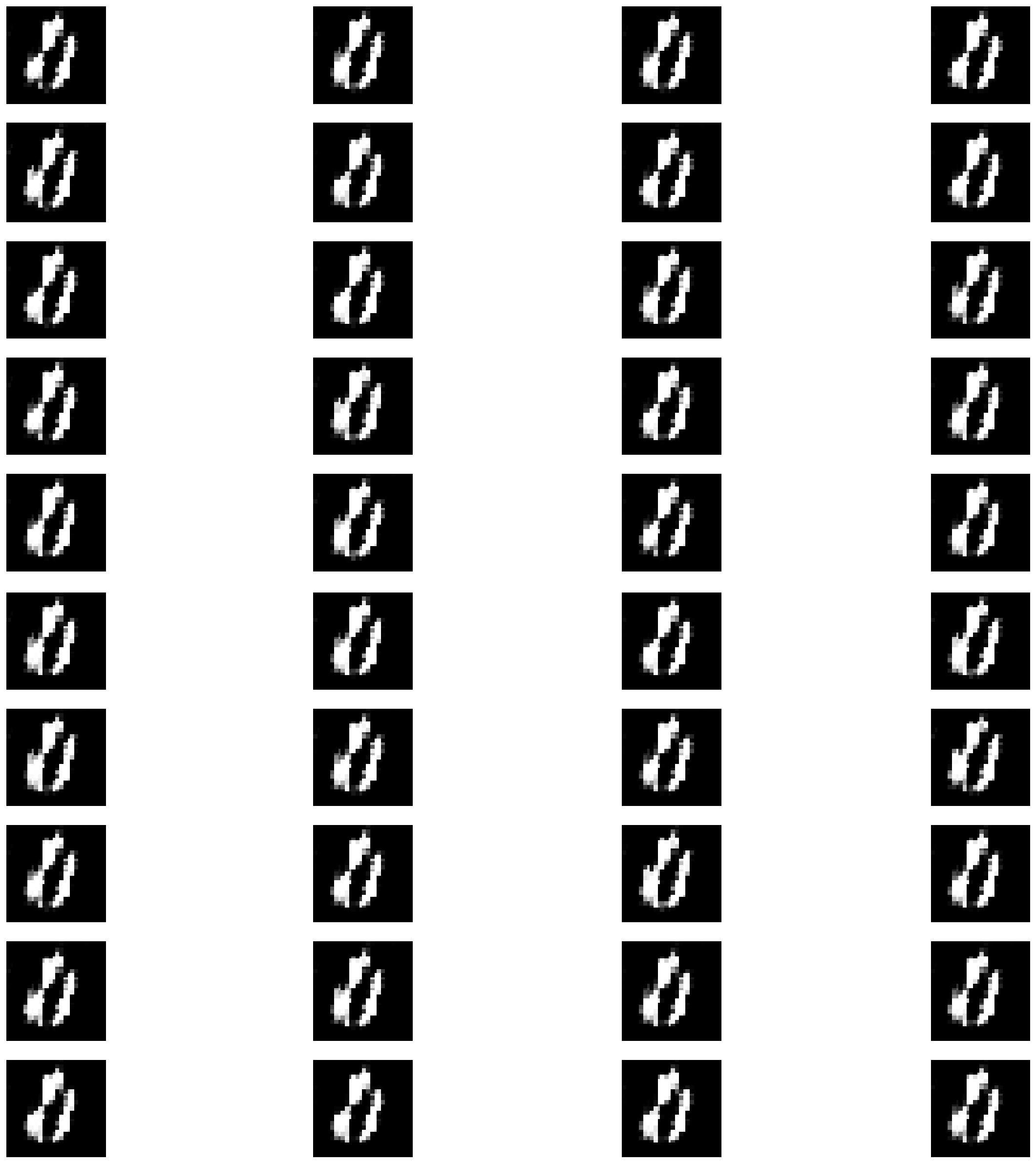


```
In [58]: generator = load_model('gan_mnist.h5')

In [57]:
```

```
      File "<ipython-input-57-7be644392e35>", line 1
    test_generatorgenerat)
                          ^
SyntaxError: invalid character in identifier
```

```
In [62]: #output some images generated by generator, with input as random noise
         noise_input=np.random.uniform(-1.0,1.0,size=[40,25])
         plot_imag(generator,noise_input,show=True,model_name='test_generator')
```



```
In [ ]:
```