HW1_BigGAN

October 18, 2019

1 Part 1: BigGAN

Setting up the environment and some helper functions

[1]:
```
import os
import tensorflow as tf
import tensorflow_hub as hub
import IPython.display
import numpy as np
import PIL.Image
from scipy.stats import truncnorm
import matplotlib.pyplot as plt
from ipywidgets import interact, interactive, fixed, interact_manual
import ipywidgets as widgets
```

[2]:
```
def interpolate(A, B, num_interps):
    alphas = np.linspace(0, 1, num_interps)
    return np.array([(1-a)*A + a*B for a in alphas])

def imgrid(imarray, cols=5, pad=1):
    if isinstance(imarray, np.ndarray):
        N, H, W, C = imarray.shape
    if isinstance(imarray, list):
        N = len(imarray)
    rows = N // cols + int(N % cols != 0)
    for i in range(N):
        plt.subplot(rows, cols, i+1)
        plt.title("%d"%i)
        plt.imshow(imarray[i])
        plt.axis('off')

def load_image(image_url, image_size=(256, 256), preserve_aspect_ratio=True):
    """Loads and preprocesses images."""
    # Cache image file locally.
    image_path = tf.keras.utils.get_file(os.path.basename(image_url)[-128:],␣
 ↪image_url)
    # Load and convert to float32 numpy array, add batch dimension, and␣
 ↪normalize to range [0, 1].
```

```
    img = plt.imread(image_path).astype(np.float32)[np.newaxis, ...] / 255.
    if img.shape[-1] == 4:
        img = img[..., :3] / tf.expand_dims(img[..., 3], -1) # pre multiply␣
 ↪alpha

    if image_size[0] != -1:
        img = tf.image.resize(img, image_size, preserve_aspect_ratio=True)
    return img
```

[3]:
```
# BigGAN-deep models
# module_path = 'https://tfhub.dev/deepmind/biggan-deep-128/1'  # 128x128␣
 ↪BigGAN-deep
module_path = 'https://tfhub.dev/deepmind/biggan-deep-256/1'  # 256x256␣
 ↪BigGAN-deep
# module_path = 'https://tfhub.dev/deepmind/biggan-deep-512/1'  # 512x512␣
 ↪BigGAN-deep

# BigGAN (original) models
# module_path = 'https://tfhub.dev/deepmind/biggan-128/2'  # 128x128 BigGAN
# module_path = 'https://tfhub.dev/deepmind/biggan-256/2'  # 256x256 BigGAN
# module_path = 'https://tfhub.dev/deepmind/biggan-512/2'  # 512x512 BigGAN
```

One very good resource for pretrained ML models is Tensorflow Hub: http://tfhub.dev

They have a convenient API for loading many ready models, and running them is extremely simple: (this cell may take a while since it's downloading a big model)

[4]:
```
tf.reset_default_graph()
print('Loading BigGAN module from:', module_path)
module = hub.Module(module_path)
```

```
Loading BigGAN module from: https://tfhub.dev/deepmind/biggan-deep-256/1
```

[7]:
```
inputs = {k: tf.placeholder(v.dtype, v.get_shape().as_list(), k)
          for k, v in module.get_input_info_dict().items()}
output = module(inputs)
```

Remember the GAN architecture:

You provide random values ("nosie") for the latent space, which the GAN had learned to turn into realistic images, w.r.t the distribution of the outputs, e.g. the visual space of "cats".

BigGAN is a conditional GAN, meaning, not only you provide random values as "noise" you also provide the class (e.g. "cat"), becasue the GAN was trained to condition the random output on that value.

[8]:
```
input_z = inputs['z']
input_y = inputs['y']
input_trunc = inputs['truncation']

dim_z = input_z.shape.as_list()[1]
vocab_size = input_y.shape.as_list()[1]
```

[9]:
```
def truncated_z_sample(batch_size, truncation=1., seed=None):
    state = None if seed is None else np.random.RandomState(seed)
    values = truncnorm.rvs(-2, 2, size=(batch_size, dim_z), random_state=state)
    return truncation * values

def one_hot(i, vocab):
    return np.eye(vocab)[np.asarray(i,np.int32).reshape(-1)]

def sample(sess, noise, label, truncation=1., batch_size=8,
           vocab_size=vocab_size):
    noise = np.asarray(noise)
    label = np.asarray(label)
    num = noise.shape[0]
    if len(label.shape) == 0:
        label = np.asarray([label] * num)
    if len(label.shape) == 1:
        label = one_hot(label, vocab_size)
    ims = []
    for batch_start in range(0, num, batch_size):
        s = slice(batch_start, min(num, batch_start + batch_size))
        feed_dict = {input_z: noise[s], input_y: label[s], input_trunc:␣
 ↪truncation}
        ims.append(sess.run(output, feed_dict=feed_dict))
    ims = np.concatenate(ims, axis=0)
    ims = np.clip(((ims + 1) / 2.0) * 256, 0, 255)
    ims = np.uint8(ims)
    return ims
```

[10]:
```
initializer = tf.global_variables_initializer()
sess = tf.Session()
sess.run(initializer)
```

[12]:
```
```

```
num_samples_w = widgets.IntSlider(value=10, min=1, max=20, step=1,␣
 ↪description='Number of samples')
truncation_w = widgets.FloatSlider(value=0.4, min=0.02, max=1, step=0.02,␣
 ↪description='Truncation')
noise_seed_w = widgets.IntSlider(value=0, min=0, max=100, step=1,␣
 ↪description='Noise seed')
display(category_w)
display(num_samples_w)
display(truncation_w)
display(noise_seed_w)
```

```
Dropdown(description='Category', options=('0) tench, Tinca tinca', '1) goldfish, Carassius aura
```

```
IntSlider(value=10, description='Number of samples', max=20, min=1)
```

```
FloatSlider(value=0.4, description='Truncation', max=1.0, min=0.02, step=0.02)
```

```
IntSlider(value=0, description='Noise seed')
```

2 Task1:

2.0.1 Select 2 other categories and generate samples from them

[13]:
```
z = truncated_z_sample(num_samples_w.value, truncation_w.value, noise_seed_w.
 ↪value)
y = int(category_w.value.split(')')[0])

ims = sample(sess, z, y, truncation=truncation_w.value)
plt.figure(figsize=(20,10))
imgrid(ims, cols=min(num_samples_w.value, 5))
```

[14]:
```
def interpolate_and_shape(A, B, num_interps):
    interps = interpolate(A, B, num_interps)
    return (interps.transpose(1, 0, *range(2, len(interps.shape)))
                   .reshape(num_samples * num_interps, *interps.shape[2:]))
```

2.0.2 Interpolations

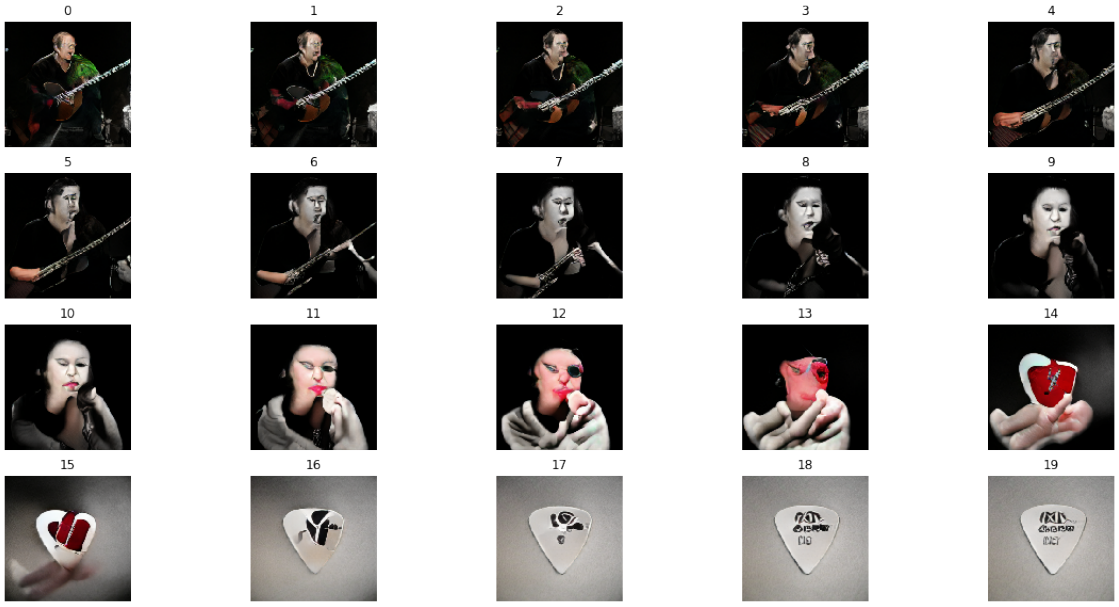
3 Task2:

3.0.1 Select 2 candidates for interpolation and create an interpolation

[25]:
```
num_samples = 1
num_interps = 20
truncation = 0.2
noise_seed_A = 0
noise_seed_B = 0
#category_A = 567 # Golden retriever
#category_B = 8 # Hen

category_A = 402 # Acoustic Guitar
category_B = 987 # corn (This was also a really trippy experiment)
category_B = 714 # pick


z_A, z_B = [truncated_z_sample(num_samples, truncation, noise_seed)
            for noise_seed in [noise_seed_A, noise_seed_B]]
y_A, y_B = [one_hot([category] * num_samples, vocab_size)
            for category in [category_A, category_B]]

z_interp = interpolate_and_shape(z_A, z_B, num_interps)
```

```
y_interp = interpolate_and_shape(y_A, y_B, num_interps)
```

[26]:
```
ims = sample(sess, z_interp, y_interp, truncation=truncation)
plt.figure(figsize=(20,10))
imgrid(ims, cols=min(num_interps,5))
```



[ ]: