GANSynth

2019年9月7日 8:40

Summary

1. interpolation
   这里的midi到audio生成，每隔固定时间段随机更换乐器随机latent vector。更换乐器以及同乐器不同pitch的变化过程平滑，应用球面插值计算laten vector。插值是generator的输入。
2. GANSynth之外还需要什么
   并不是完成Video-Music representation就可以解决问题，GANSynth还是需要得到具体的旋律音符序列，所以如果想要用现有的generator，multi-task设计依旧需要完成具体旋律的生成。
3. progressive growing of GAN
   a new methodology for training GAN, which indicates to grow both G and D progressively: starting from a low resolution.

Midi & Pitch Estimation Test

- Midi Test: interpolate midi notes
  - test.mid -> test.wav
  - test_2.mid -> test_2.wav
- Pitch Estimation Test: estimate & get pitches first, interpolate notes then
  test.webm with a threshold of confidence for pitch estimation
  a degeneration version for single track transcription here, actually
  - pe_00.wav: confidence >= .0
  - pe_05.wav: confidence >= .5
  - pe_08.wav: confidence >= .8

```
Generating 1764 samples...
generate_samples: generated 1764 samples in 458.7194254398346s
Saved to C:\Users\Gao\Documents\mc\magenta\magenta\models\gansynth\output\generated_clip.wav
```

```python
# If a MIDI file is provided, synthesize interpolations across the clip
unused_ns, notes = gu.load_midi(FLAGS.midi_file)
# Distribute latent vectors linearly in time
z_instruments, t_instruments = gu.get_random_instruments(
    model,
    notes['end_times'][-1],
    secs_per_instrument=FLAGS.secs_per_instrument)
# Get latent vectors for each note
z_notes = gu.get_z_notes(notes['start_times'], z_instruments, t_instruments)
# Generate audio for each note
print('Generating {} samples...'.format(len(z_notes)))
audio_notes = model.generate_samples_from_z(z_notes, notes['pitches'])
```

```python
# Make a single audio clip
audio_clip = gu.combine_notes(audio_notes,
                              notes['start_times'],
                              notes['end_times'],
                              notes['velocities'])
```

```
{'pitches': array([65, 74, 70, ..., 46, 39, 55]), 'velocities': array([64, 64, 64, ..., 64, 64, 64]), 'start_times': array([  0. ,   0. ,
0. , ..., 148.5, 148.5, 148.5]), 'end_times': array([  0.75,   0.75,   0.75, ..., 150.  , 150.  , 150.  ])}
```

awqqqsqwswe

instruments 2D-array

```
[[ 1.52508152  0.83017978  0.56770398 ...  0.88009917 -0.38660623
   0.87204893]
 [ 0.59832578  1.06523886 -1.08107413 ... -1.38369011  0.93347232
   1.61114299]
 [-0.90956932  1.7075086  -0.2497775  ...  0.19378978 -0.87584415
   0.47417045]
 ...
 [ 1.37885094  0.35699163  1.19863052 ...  0.24354391  0.50356033
  -0.49411357]
 [-0.52399878  0.03027794  0.05174636 ... -1.17639471 -0.48137524
  -0.73951683]
 [-1.36955472 -0.13538165 -0.09518699 ... -1.01936217  0.47086826
  -0.69294608]]
[-1.00000000e-04  6.24990417e+00  1.24999083e+01  1.87499125e+01
  2.49999167e+01  3.12499208e+01  3.74999250e+01  4.37499292e+01
  4.99999333e+01  5.62499375e+01  6.24999417e+01  6.87499458e+01
  7.49999500e+01  8.12499542e+01  8.74999583e+01  9.37499625e+01
  9.99999667e+01  1.06249971e+02  1.12499975e+02  1.18749979e+02
  1.24999983e+02  1.31249988e+02  1.37499992e+02  1.43749996e+02
  1.50000000e+02]
```

gu.load_midi

- [min_pitch, max_pitch]: best pitch range for human hearing
- return midi sequences & note labels

gu.get_random_instruments

- n_instruments: number of instruments needed, switch to another instrument randomly after specified seconds indicated by *secs_per_instrument*
- z_instruments: latent vectors randomly generated for each instrument
- t_instruments: time frame generated as arithmetic progression for each instrument

gu.get_z_notes

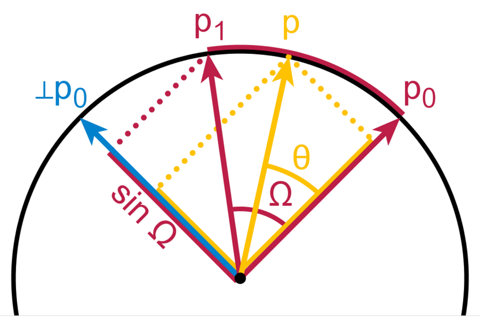
- Global conditioning on latent and pitch vectors allow GANs to generate perceptually smooth interpolation in timbre, and consistent timbral identity across pitch. (e.g. here we transform from one instrument to another in a specified time interval, thus it needs a smooth transition perceptually)
- here we compute a slerp vector using a time proportion as parameter for one note stided intervals

$$slerp = \frac{\sin[(1-t)\Omega]}{\sin\Omega} p_0 + \frac{\sin[t\Omega]}{\sin\Omega} p_1$$

```
def get_z_notes(start_times, z_instruments, t_instruments):
    """Get interpolated latent vectors for each note."""
    z_notes = []
    for t in start_times:
        idx = np.searchsorted(t_instruments, t, side='left') - 1
        t_left = t_instruments[idx]
        t_right = t_instruments[idx + 1]
        interp = (t - t_left) / (t_right - t_left)
        z_notes.append(
            slerp(z_instruments[idx], z_instruments[idx + 1], interp))
    z_notes = np.vstack(z_notes)
    return z_notes
```

interp $\in [0,1]$

- spherical linear interpolation, slerp
- p0: first poin, p1: last point, t: paramter



model.generate_samples_from_z

here interpolated vectors are used to generate batched samples
samples stored as an array and returned

Args:

- z: lantent vectors
- pitches: array of pitches (returned by load_midi)

```
    labels = self._pitches_to_labels(pitches)
    n_samples = len(labels)
    num_batches = int(np.ceil(float(n_samples) / self.batch_size))
    n_tot = num_batches * self.batch_size
    padding = n_tot - n_samples
    # Pads zeros to make batches even batch size.
    labels = labels + [0] * padding
    z = np.concatenate([z, np.zeros([padding, z.shape[1]])], axis=0)
    # Generate waves
    start_time = time.time()
    waves_list = []
    for i in range(num_batches):
      start = i * self.batch_size
      end = (i + 1) * self.batch_size
      waves = self.sess.run(self.fake_waves_ph,
                            feed_dict={self.labels_ph: labels[start:end],
                                       self.noises_ph: z[start:end]})
      # Trim waves
      for wave in waves:
        waves_list.append(wave[:max_audio_length, 0])
    # Remove waves corresponding to the padded zeros.
    result = np.stack(waves_list[:n_samples], axis=0)
    print('generate_samples: generated {} samples in {}s'.format(
        n_samples, time.time() - start_time))
    return result
```

gu.combine_notes

simply merged sampels generated in last step into a wav file

auxiliary loss function

laten vector size hearing range

- first concat(noise, one_hot_pitch) which is a (8, 256) + (8, 61) shape size, 8: batch size
- fake_data_ph: gernerator output, not a wavefrom actually in this step
- fake_waves_ph: convert fake_data_ph into sample waves

```
# (label_ph, noise_ph) -> fake_wave_ph
labels_ph = tf.placeholder(tf.int32, [batch_size])
noises_ph = tf.placeholder(tf.float32, [batch_size,
                                        config['latent_vector_size']])
num_pitches = len(pitch_counts)
one_hot_labels_ph = tf.one_hot(labels_ph, num_pitches)
with load_scope:
  fake_data_ph, _ = g_fn((noises_ph, one_hot_labels_ph))
  fake_waves_ph = data_helper.data_to_waves(fake_data_ph)
```

data_helper.data_to_waves

spectrams -> STFT -> wave

```
def specgrams_to_waves(self, specgrams):
  """Converts spectrograms to stfts."""
  return self.stfts_to_waves(self.specgrams_to_stfts(specgrams))
```