Build a Text-To-Speech system

Google AI ML Winter Camp, SHA, Jan 14 -- 18, 2019

Description

In this project, you will build a speech synthesis system using state-of-the-art techniques.

The speech synthesis system has two parts:

- Go from characters to magnitude spectrogram (char2feats)
- Go from spectrogram to actual audio (vocoder = phase reconstruction & super-resolution), called WaveRNN.

A team can tackle each task, or you can divide your team into two. At the end, the teams can combine their results to produce a full end-to-end system. Both parts can be implemented independently.
If you finish early, take a look at the "Bonus" task, where you would start building a real-time demo server.

Content

Foreword:
Remember to use `py_utils.HasShape`, and use the google documentation style. Shapes must always be documented. Also, write tests which check the shapes. It's also useful to use `py_utils.CheckNumerics` every so often. Write unit tests for all modules. Add summaries for waveforms, spectrograms, with utterance IDs so that you can go back and listen to them.

Day 1: Data preparation
Download and run the lingvo toolkit. Try the WMT16 system (follow the instructions), just to get warmed up.

Download the LJ dataset. Parameterize for:

- char2feats
- waveRNN

Create a directory called `lingvo/tasks/tts`. You will create a set of tools similar to the ASR data preparation.

Remember to install `sox`. You will need to create a tool similar to the ASR one.

For char2feats, you will create `tfrecords` of `tf.Example` which contain the following:

- `uttid`: utterance ID. This will be useful for debugging. We will show this in the tensorboard summaries, a string.
- `transcript`: the text sentences, a string.
- `frames`: the spectrograms, a float tensor.
- Optionally, you could pre-tokenize the transcript into a tensor of integer IDs.

Note: there is a "bug" in the current feature extractor. You cannot change the context/stride. Just extract the frames and use `tf.reshape(frames, [b, -1, 80])`. (See ASR code.) The char2feats model will tokenize the `transcript` into graphemes (characters).

The WaveRNN needs to map from spectrograms to audio samples:

- `uttid`: utterance ID. Always keep that for debugging.
- `frames`: as above, the spectrograms.
- `audio_samples`: 24kHz audio samples, 0-1 floats.
- For debugging, write the associated text as `transcript`.

Write the `input_generator` for each. Run it in isolation to verify that you get what you expect, just like we did for the ASR input generator. Make sure that the scale of the input samples is done correctly, otherwise the energies will be very small.

Bonus: write a utility function that scans that tf records to find an audio sample and plays out the audio, for a given uttid.

Char2feats:

- Create encoder (stacked LSTMs)
- Create decoder (attention + stacked LSTMs)
- Create loss

You need to read the paper carefully and make sure the model converges. You can start with the machine translation (MT) encoder. A reference implementation is here. A good system diagram is here.

Bonus: Can you try the Transformer encoder? Does it lead to better quality?

WaveRNN:

- Convert to 12kHz, 8-bit samples for development. This will train faster, with, of course, a loss in quality
- Create network & dual softmax loss
- Bonus: Create sparse network

The encoding is relatively straightforward. The difficulty here is that it's hard to train.

Bonus:

To make a fully functional system, you'll need:

- Create the Inference Graph
- Efficient CPU implementation for sparse networks
- Demo / serving

It will be hard to have a real-time demo, but you can start work towards that goal during the course, and finish later.

References

[1] Tacotron2

[2] WaveRNN