Manual for tc2 envt: text-to-multimodal speech synthesis using Tacotron2

Author: Gérard BAILLY
Contributors:

The tc2 model maps input symbols (text characters or phones) to one or several sequences of parameters via a text encoder, autoregressive attention maps - that perform text-to-sequence alignment by activating input embeddings according to previously predicted parameters via a prenet – and decoders that perform parameter and end-of-sequence (EoS) prediction.

Speaker and style embeddings can be trained and added to all embeddings at the output of the text encoder.

A phonetic predictor can also be added that predicts phones from the output of the text encoder

A. Specification of data and model parameters

Most of the data specification, model parameters and default training and test material are set in a .yaml file. Some keys strongly depend on the number of decoders `nd`:

- Keys "dir_data", "ext_data", "dim_data" , "fe_data", "use_postnet" , "n_frames_per_step", "decoder_rnn_dim", "prenet_dim", "gate_threshold", "p_prenet_dropout", "p_postnet_dropout", "p_attention_dropout", "p_decoder_dropout", "p_teacher_forcing", "attention_rnn_dim", "attention_dim", "attention_location_n_filters", "attention_location_kernel_size", "postnet_embedding_dim", "postnet_kernel_size" , "postnet_n_convolutions" , "factor_gate" are lists of dimension `nd`. Note that the use of postnets is optional via the Boolean list key "use_postnet" and that tc2 offers the possibility for decoders to predict n_frames_per_step > 1

B. Data

I.Input data

Licit input symbols (letters, punctuations, phones, markers, etc) for the language (declared in the key "language" in the .yaml) are listed in `def_symbols.py`:

- valid_symbols lists the language-specific input phones. Note that input phones should be prefixed by "@" (e.g "Je suis @b@j@e~") in the input text or surrounded by braces "{" "}" (e.g "Je suis {b j e~}")
- _specific_characters lists language-specific letters (e.g. accented characters…). Note that we use "§" to indicate beginning and end of paragraphs

II.Output data

Parameter sequences are described in the .yaml file by keys ”dir_data”, ”ext_data”, ”dim_data” , ”fe_data”. Sequences should be stored in a simple custom binary files as float32 frames dir_data/*.ext_data in the following format:

- A header of 4 int32 integers specifying: (1) the number of frames; (2) the number of parameters per frame (should be equal to ”dim_data”); (3) the numerator n and (4) the denominator d for calculating the frame rate n/d that should be equal to ”fe_data” (enable exact rational rate when lag is expressed in number of samples, e.g. 22050/256 = 86,132812… for WAVEGLOW or HifiGAN mel-spectrograms).
- Followed by frames of length ”dim_data”, one after the other

We recommend to name files with the following format:
<author>_<book>_<speaker>_<style>_<volume>_<chaper>.”ext_data”

with <speaker> and <style> resp. included in the “speakers” and “styles” key lists in the .yaml, so that to automatically bias the outputs of the text encoder with the appropriate speaker and style embeddings if keys “nb_speakers”>0 and “nb_styles”>0 respectively.

Aligned phonetic transcripts can be also given to train a phonetic decoder with a list of a language-specific output phones listed in `def_symbols.py` as `out_symbols` (that equals `valid_symbols` plus diphones or silences if necessary.

To train a phonetic decoder, you have to set the key “factor_pho” to a non-zero float: e.g. 1.00

III.Multiple decoders

Keys concerning output data (”dir_data”, ”ext_data”, ”dim_data” , ”fe_data”), dimensions of attention mechanism are lists. They shape decoders.

There are three ways to add decoders:

1. Joint decoder: simply concatenate frames and simply add dimensions in “dim_data”. Of course their sampling frequencies should be equal.
2. Add another regressor at the output of another decoder (often the first audio one). Their sampling frequencies should be equal in ”fe_data” and corresponding values in “decoder_rnn_dim” should be equal. The value of “prenet_dim” for that additional decoder should be negative or zero: its absolute value indicates the index of the decoder it phagocytes.
3. Add an additional plain decoder plugged at the output of the text encoder. It can operate with different dimensions, notably with a different ”fe_data” and ” n_frames_per_step”. However, at synthesis, the gate of the first decoder determines the end of the sequence.

IV.Alignments

Training and test text/sequences are provided by .csv files that may contain different descriptions of utterances as fields separated by the “|” delimiter:

- <file_name>|<start_ms>|<end_ms>|<input text>(|<aligned output phones>)
- LEX|<input text>|<aligned output phones>

The first description is the standard one. <input text> may mix orthographic and phonetic input. <aligned output phones> is optional. The second description is used to use aligned lexicons, quite interesting to cope with unusual or contemporary spellings rarely encountered in conversation or free-of-rights audiobooks.

We recommend to include 0.13s of silence ("lgs_sil_sides" key in the .yaml[1]) before and after all utterances, so that to ease tc2 in "explaining" start and end punctuations such as quotes, parenthesis, etc. When utterances have no start punctuations such as quotes or "§", we use the last punctuation of the previous utterance if in a paragraph or use "," as a default. Same for the end punctuation.

C. Training

Model training is performed by `do_train.py` (the .csv file to be processed is specified in .yaml by the key "nm_csv_test") Optional parameters are listed below:

- -o <dir_name> : directory to save checkpoints after each iteration of the training files
- -c <file _name> : loading pre-trained model. Note that tc2 adds text, speaker, style and output phone embeddings if the pre-trained model has less embeddings than the model to be trained
- --model_name <file _name> : prefix of checkpoints
- --hparams <string>: comma separated "name=value" pairs to override the .yaml values
- --freeze <string>: freeze units by regular expression
- --silent: run silently
- --config <file _name>: .yaml configuration file
- --id_new_speaker <int>: id of the embeddings of the speaker to copy from for a new speaker
- --stateful: use end state of encoder lstm as initial state of the next utterance for chaining utterances

D. Testing

Batch testing is performed by `do_syn.py` (the .csv file to be processed is specified in .yaml by the key "nm_csv_train") and interactive tts by `do_tts.py`. Optional parameters are listed below:

- --output_directory: directory to save generated files
- --config: .yaml configuration file

---

[1] Note that the "lgs_sil_add" key in the .yaml (by default .1s) is used to have robust end-of-sequence (EoS) prediction: these frames of ground-truth silence are added at the end of each utterance with corresponding gate targets set to "True".

- --no_auto_numbering: no adding of "_{:04d}" at the end of filenames
- -p or –prediction: prediction instead of synthesis
- --play_wav: play sound
- --draw: draw attention maps
- --overwrite: overwrite existing files if any
- -g or –ground_truth: generate ground-truth audio and parameter files for all utterances (with extensions "_org" at the end of filenames)
- --parameter_files: generate parameter files with proprietary format
- -t or –tacotron: name of the Tacotron model
- --save_embeddings: Save output embeddings of text-encoder and text, speaker and style embeddings in .mat file
- -v or -vocoder: filename of the vocoder (only "hifigan" and "waveglow" in the filename for now)
- --speaker: speaker name if multiple speakers (override speaker name in the original filename of the original sequence if any)
- --style: style name if multiple styles (override style name in the original filename of the original sequence if any)
- -r or –sampling_rate: sampling rate of the generated audio (default: 22050)
- --hparams <string>: comma separated "name=value" pairs to override the .yaml values
- --phonetic_only: output only phonetic predictions if phonetizer is on
- --silent: run silently
- --stateful: use end state of encoder lstm as initial state of the next utterance for chaining utterances (if the model has been trained accordingly)

Please note that `do_syn.py` breaks text (in the input .csv file) at "§" characters: It will add a "short_pause" (default: 0.15s) silence at these positions or a "long_pause" (default: 0.45s) if the character is doubled i.e. "§§", indicating a paragraph break. If stateful option is set, the initial state of the next utterance is reset to zero). This silence is picked up in the audiofile with ambient silence named "sil_<speaker_name>_<sampling_rate>.wav" if it exists otherwise set to zeros.

E. Finetuning Hifigan

I. Download hifi-gan-master directory

From the original github from Kakao Enterprise[2], the original file `meldataset.py` has been modified to read the custom format of our mel-spectrograms files generated by `do_syn.py` i.e.:

```
(lg_data,dim,num,den) = tuple(np.fromfile(nm_mel, offset=0, count=4,
  dtype=np.int32)); fe=num/den;
mel=torch.from_numpy(np.reshape(np.fromfile(nm_mel, offset=16,
  count=lg_data*dim, dtype=np.float32),(lg_data,dim)).transpose())
```

II. Generating train and validation sets

The script below generates at most 500 utterances from each training speaker (picked in ALL.csv and stored in train_hifigan.csv).

Then `do_syn.py` generates `_syn_WAVEGLOW/*prd.WAVEGLOW` vs `_syn_WAVEGLOW/*org.wav` using options `-g -p --parameter_files` (resp. groundtruth, prediction mode and storing parameter files)

Finally, 100 generated files are used as validation data while the rest is used as training data (resp. stored in `hifi-gan-master/validation.csv` and `hifi-gan-master/train.csv`

```
lst_spk=`grep "speakers: \[" tc2.yaml | sed -e "s/', '/ /g" -e "s/']//" -e
  "s/speakers: \['//"`; a_spk=(${lst_spk}); nb_spk=${#a_spk[@]};

>train_hifigan.csv
for i in ${!a_spk[*]}; do
  grep -e "_${a_spk[$i]}_" ALL.csv| grep -v LEX | shuf |head -500 >>
  train_hifigan.csv
done
mod=tacotron2_ALL;
do_syn.py \
  --tacotron $mod \
  --config tc2.yaml -g -p --parameter_files \
  --hparams "{dim_data: [80], nm_csv_test: 'train_hifigan.csv'}"
produit _syn_WAVEGLOW/*prd.WAVEGLOW vs _syn_WAVEGLOW/*org.wav
/bin/rm _syn_WAVEGLOW/*_prd.wav
awk -F '|' '{printf("%s_%04d|%s|%s\n",$1,n,$4,$4); n=n+1;}'
  train_hifigan.csv > train.csv
shuf train.csv > /tmp/train; mv /tmp/train train.csv
head -100 train.csv > hifi-gan-master/validation.csv
tail -n +101 train.csv > hifi-gan-master/train.csv
```

[2] https://github.com/jik876/hifi-gan

III. Finetune hifigan generator & discriminator

Change directory to `hifi-gan-master`. The script below fills the `wavs` and `mels` directories resp. with original audio and predicted mel spectrograms.
It hen just runs the train.py script with `--fine_tuning` option set to true. At each epoch, two files are stored in the `cp_hifigan` directory: the generator `g_{iteration number}` and the discriminator `do_{iteration number}. g_{iteration number}` is the file that can be used as fine-tuned hifigan vocoder by `do_syn.py`

```
for f in `ls ../_syn_WAVEGLOW/*org.wav`; do
nm="${f##*/}"; nm="${nm%%_org.*}";
ln -f -s ../$f wavs/$nm.wav
ln -f -s ../../_syn_WAVEGLOW/${nm}_prd.WAVEGLOW mels/$nm.mel
done
mkdir cp_hifigan
train.py --fine_tuning True --config config_v1.json \
  --input_wavs_dir wavs --input_mels_dir mels \
  --input_training_file train.csv --input_validation_file validation.csv
```

F. Other resources

The module `load_csv.py` loads csv files

The script `do_process_files.py` computes mel spectrograms from 22050Hz audiofiles and store then in custom binary files (see format in §B.II).