

Report on Tacotron-2

Mei

Structure of the report:

1. The process of my work
 - 1.1 outline of my working process
 - 1.2 basic difference between tacotron and tacotron-2
2. Introduction on the Tacotron-2 structure
 - 2.1 tacotron-2 structure based on source code I debug
 - 2.2 tacotron-2 structure based on the working process(This part is been deleted, since I find it repeats the work in 2.1, but I will create it in detailed version in the future)
3. Compare the paper_hparams.py and hparams.py

1. The process of my work

In this part, I will focus on how I completed this assignment step by step according to the requirements of the instruction book, and report on some basic outputs. As for the detailed results and analysis of the results. I will combine the Tacotron-2 structure introduction and some source code descriptions for displaying in the second part——introduction on the Tacotron-2 structure.

1.1. Outline of my working process

- Firstly I followed the instructions and loaded the python module and activate additional module, using:

```
$ module load Python/3.6.4-foss-2018a
$ module load alsa-lib/1.1.9-GCCcore-6.4.0
$ module load libav/12.3-GCCcore-6.4.0
$ module load PortAudio/19.6.0-GCCcore-6.4.0
$ module load TensorFlow/1.12.0-foss-2018a-Python-3.6.4
$ module load h5py/2.7.1-foss-2018a-Python-3.6.4
$ module load numba/0.37.0-foss-2018a-Python-3.6.4
```


For the new version of requirement depended for librosa(resampy), I loaded the required libraries:

```
$ pip install --user resampy==0.2.2
```





- Then install all the requirement dependencies by installing the requirement.txt:

```
$ pip install --user -r requirements.txt
```

- Downloaded the LJSpeech corpus and untar the dataset
`$ wget https://data.keithito.com/data/speech/LJSpeech-1.1.tar.bz2`
`$ tar -xvjf LJSpeech-1.1.tar.bz2`
- Then preprocessed the data
`$ nano preprocess_lj_data.sh`(the content of preprocess_lj_data.sh is inside the instruction)
`$ sbatch preprocess_lj_data.sh`
 It will appear after the preprocessing is completed, the training_data folder

 **training_data**

Contains the following files:

 **audio**
 **linear**
 **mels**
 **train.txt**

These are the features for training, and here is log piece

```
Write 13100 utterances, 6858136 mel frames, 1885987400 audio timesteps, (23.76 hours)
Max input length (text chars): 187
Max mel frames length: 809
Max audio timesteps length: 222475
```

- Trained the job
`$ nano training.sh`
`$ sbatch training.sh`
 In this process, I made so many mistakes.
 Firstly, when I `$ sbatch training.sh` try to submit the training job, it said that the node number should be 1 instead of 8, this is different from the instruction, and I am still working on the reason.
 Then I have troubles in timeout, and I found that I set the training hour for only 8 hours, this is obviously out of time, one example for my trouble could be shown like this:

```

Job ID           : 28780308
Name             : training_LJSpeech
User            : s5075580
Partition       : regular
Nodes           : pg-node205
Number of Nodes : 1
Cores           : 1
Number of Tasks : 1
State           : TIMEOUT,CANCELLED
Submit          : 2023-03-08T08:01:21
Start           : 2023-03-08T08:01:22
End             : 2023-03-08T16:01:52
Reserved walltime : 08:00:00
Used walltime    : 08:00:30
Used CPU time    : 07:57:15 (efficiency: 99.32%)
% User (Computation): 98.64%
% System (I/O)   : 1.36%
Mem reserved     : 30G
Max Mem (Node/step) : 25.13G (pg-node205, per node)
Full Max Mem usage : 25.13G
Total Disk Read  : 6.95M
Total Disk Write : 199.01K

```

In the last try of training, I set the steps to 500 and the training hour to 15, and finished it.




```

100%|██████████| 13085/13100 [11:46:01<00:51, 3.44s/it]
100%|██████████| 13086/13100 [11:46:03<00:39, 2.82s/it]
100%|██████████| 13087/13100 [11:46:07<00:42, 3.30s/it]
100%|██████████| 13088/13100 [11:46:09<00:35, 2.93s/it]
100%|██████████| 13089/13100 [11:46:14<00:38, 3.46s/it]
100%|██████████| 13090/13100 [11:46:18<00:37, 3.71s/it]
100%|██████████| 13091/13100 [11:46:21<00:29, 3.33s/it]
100%|██████████| 13092/13100 [11:46:24<00:26, 3.33s/it]
100%|██████████| 13093/13100 [11:46:27<00:23, 3.39s/it]
100%|██████████| 13094/13100 [11:46:31<00:20, 3.47s/it]
100%|██████████| 13095/13100 [11:46:34<00:16, 3.22s/it]
100%|██████████| 13096/13100 [11:46:36<00:11, 2.89s/it]
100%|██████████| 13097/13100 [11:46:40<00:10, 3.37s/it]
100%|██████████| 13098/13100 [11:46:45<00:07, 3.62s/it]
100%|██████████| 13099/13100 [11:46:49<00:03, 3.83s/it]
100%|██████████| 13100/13100 [11:46:53<00:00, 3.98s/it]
#####

```

Tacotron GTA Synthesis

But the state of it is still failed, I guess it can also explain why the synthesizing process also failed, I would go into the details of it later.

 `wavenet_model.ckpt-50.data-00000-...`
 `wavenet_model.ckpt-50.index`
 `wavenet_model.ckpt-50.meta`

1.2. Basic difference between Tacotron and Tacotron-2

Logically speaking, this part should belong to the Tacotron-2 structure, which is what I will introduce in the second part, but since I mentioned the difference between using the Tacotron-2 model and other models when introducing the synthesis process, I want to briefly introduce the available models and their differences in advance in this part of the workflow.

In the training process, I have different options, which I only tried the Tacotron-2 model, and it has the similar options in the synthesizing approach.

Training:

To train both models sequentially (one after the other):

```
python train.py --model='Tacotron-2'
```

Feature prediction model can separately be trained using:

```
python train.py --model='Tacotron'
```

checkpoints will be made each 5000 steps and stored under logs-Tacotron folder.

Naturally, training the wavenet separately is done by:

```
python train.py --model='WaveNet'
```

logs will be stored inside logs-Wavenet.

Note:

- If model argument is not provided, training will default to Tacotron-2 model training. (both models)
- Please refer to train arguments under [train.py](#) for a set of options you can use.
- It is now possible to make wavenet preprocessing alone using [wavenet_preprocess.py](#).

Synthesize audio in an end-to-end (text-to-audio) fashion (both models run simultaneously):

```
python synthesize.py --model='Tacotron-2'
```

For the spectrogram prediction network, there are three types of prediction results for mel spectrograms:

Evaluation (comprehensive evaluation of custom sentences). This is what we usually use when we have a complete end-to-end model.

```
python synthesize.py --model='Tacotron'
```

Natural synthesis (let the model make predictions alone by feeding the output of the last decoder into the next timestep).

```
python synthesize.py --model='Tacotron' --mode='synthesis' --GTA=False
```

Efficient alignment synthesis (default: model is forced to train with valid ground truth

labels). This synthesis method is used when predicting mel spectra for training wavenet. (As stated in the text, yields better results)
python synthesize.py --model='Tacotron' --mode='synthesis' --GTA=True

Synthesize the waveform with the previously synthesized Mel spectrum:

```
python synthesize.py --model='WaveNet'
```

This can also map the shortcoming of tacotron only using Griffin Lim as a temporary synthesizer. In tacotron-2, this problem has been solved, so when you run tacotron-2, the supporting wavenet will also start.

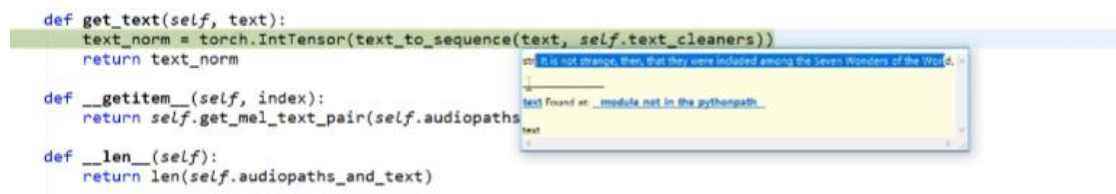
2. Introduction on the Tacotron-2 structure

In this part, I will elaborate on the structure of Tacotron-2 by observing the training process and the results produced. I will also combine the source code of Tacotron-2 (I use the debug to kind of explore the source code) and compare it with the observation of the training process to better explore the structure of Tacotron-2 in detail.

2.1. Tacotron-2 structure based on the source code I debug

- Data loader:

Text data: Firstly loading the text data

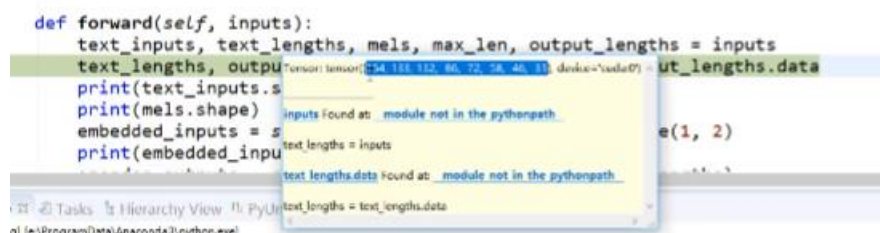


```
def get_text(self, text):
    text_norm = torch.IntTensor(text_to_sequence(text, self.text_cleaners))
    return text_norm

def __getitem__(self, index):
    return self.get_mel_text_pair(self.audiopaths

def __len__(self):
    return len(self.audiopaths_and_text)
```

Then changing each character to a number identification, for example, change a to 23 and change r to 29. You can see the number representation in the following picture.



```
def forward(self, inputs):
    text_inputs, text_lengths, mels, max_len, output_lengths = inputs
    text_lengths, output_lengths = text_lengths.data, output_lengths.data
    print(text_inputs.shape)
    print(mels.shape)
    embedded_inputs = self.embedding(text_inputs)
    print(embedded_inputs.shape)
```

Since the tacotron-2 dealing with the data in batch, for example, 8 sentences in a batch, and the length of different sentence is different, so the tacotron-2 does a work called adding zero number, it will take the max length sentence of the standard and add the zero number to other sentences until they all get the same length of the max one. You can better understand it by looking at the following picture, you can observe the

sequence last number are all zeros.

```
def forward(self, inputs):
    text_inputs, text_lengths, mels, max_len, output_lengths = inputs
    text_inputs = torch.tensor(text_inputs, dtype=torch.long, device='cuda:0')
    text_lengths = torch.tensor(text_lengths, dtype=torch.long, device='cuda:0')
    mels = torch.tensor(mels, dtype=torch.float, device='cuda:0')
    max_len = torch.tensor(max_len, dtype=torch.long, device='cuda:0')
    output_lengths = torch.tensor(output_lengths, dtype=torch.long, device='cuda:0')

    # Transpose text_inputs to (batch, seq_len, embedding_dim)
    text_inputs = text_inputs.transpose(1, 2)

    # Embedding
    embedded_text_inputs = torch.nn.LSTM(text_inputs, text_lengths, mels, max_len, output_lengths)

    # LSTM
    lstm_out, (h, c) = self.lstm(embedded_text_inputs)

    # Output
    output = self.output(lstm_out)

    return output
```

After this method, we will get the basic dimension of input text data (batch size, sequence length), I set the batch size as 8. So the dimension should be (8, sequence length).

Audio data:

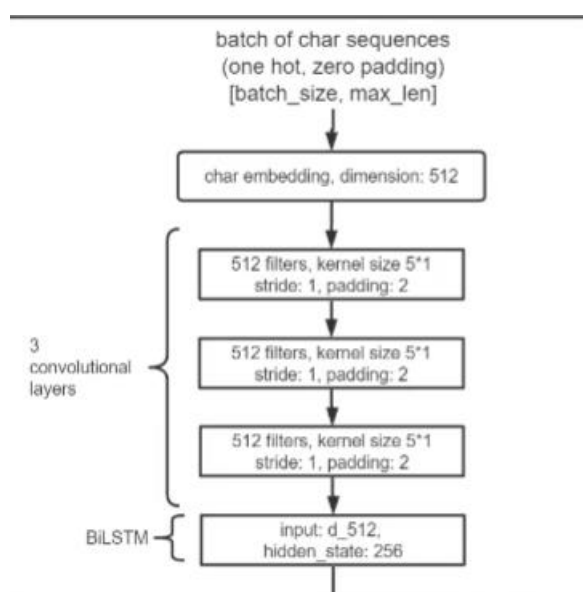
Getting speech data, reading, windowing, short-time Fourier transform, and using mel filter to extract the features(80 filters). The Fourier transform and extract mel features can be shown in the following picture.

```
assert(torch.min(y.data) >= -1)
assert(torch.max(y.data) <= 1)

magnitudes, phases = self.stft_fn.transform(y)
magnitudes = magnitudes.data
mel_output = torch.matmul(self.mel_basis, magnitudes)
mel_output = self.spectral_normalize(mel_output)
return mel_output
```

- Encoder:

After getting the text input(8, max sequence length), 8 is the batch size, encoder transform the original text input to embeddings, (8, 512, max sequence length), 512 is the features replacing the original number(but I have no idea why it changes to 512 yet), the embedding can be seen in the structure picture:



As the graph shows, after getting the embeddings, they will pass through 3 convolutional layers and one BiLSTM layer, get the hidden state. The final output dimension is [batch_size 8 , 512, max_length]

```
torch.Size([80, 318])
torch.Size([8, 154])
torch.Size([8, 154])
torch.Size([8, 154])
torch.Size([8, 80, 864])
torch.Size([8, 512, 154])
```

- Prenet

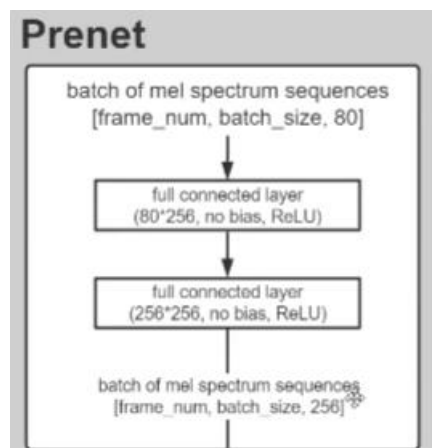
Made two linear functions to convert 80-dimensional mel features into 256 dimensions(But I still have no idea why need to convert to 256 dimension compared to 80)

```
print(decoder_input.shape)
decoder_inputs = self.parse_decoder_inputs(decoder_inputs) # teacher forcing方法, 所以直接取每帧值
print(decoder_inputs.shape)
decoder_inputs = torch.cat((decoder_input, decoder_inputs), dim=0) # 第一个位置的帧和后位置的帧拼起来, 这就全了
print(decoder_inputs.shape) # 864+1 = 865
decoder_inputs = self.prenet(decoder_inputs)
print(decoder_inputs.shape) # 865 8 256

self.initialize_decoder_states(
    memory, mask=~get_mask_from_lengths(memory_lengths))

mel_outputs, gate_outputs, alignments = [], [], []
while len(mel_outputs) < decoder_inputs.size(0) - 1:
```

```
nsolve  Tasks  Hierarchy View  PyUnit
[debug] [E:\ProgramData\Anaconda\python.exe]
:h.Size([8, 864, 80])
:h.Size([8, 864, 80])
:h.Size([864, 8, 80])
:h.Size([864, 8, 80])
:h.Size([865, 8, 80])
:h.Size([865, 8, 256])
```



- Attention and decoder

The previous decoder input (865,8,256) extracted one frame dimension should be (8,256) and splice this dimension with the attention context dimension, and the obtained dimension is (8,768)


```

368 cell_input = torch.cat((decoder_input, self.attention_context), -1)
369 print(cell_input.shape)
370 self.attention_hidden, self.attention_cell = self.attention_rnn(
371     cell_input, (self.attention_hidden, self.attention_cell)) # hidden cell 经过单元之后
372 print(self.attention_hidden.shape)
373 self.attention_hidden = F.dropout(
374     self.attention_hidden, self.p_attention_dropout, self.training)
375 print(self.attention_hidden.shape)
376
377 attention_weights_cat = torch.cat(

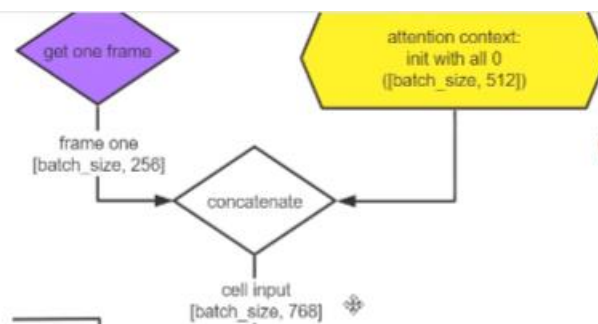
```

Console Output:

```

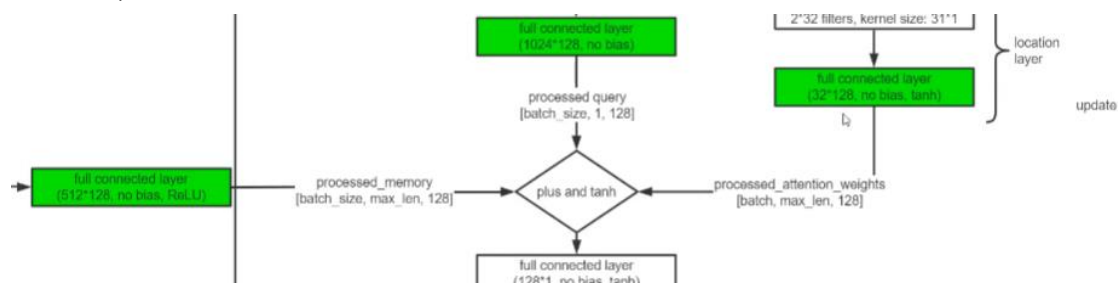
train.py [debug] [e:\ProgramData\Anaconda3\python.exe]
torch.Size([8, 154])
torch.Size([8, 154])
torch.Size([8, 512])
torch.Size([8, 154, 512])
torch.Size([8, 256])
torch.Size([8, 768])

```



The synthesized features are obtained through lstm to obtain hidden layer features (8,1024)

Accumulate attention, provide position information, decoder provide voice information, encoder provide text information



Then add three together
Weight multiplied by decoder_output.

3. Compare paper_hparams and hparams

For different hyperparameters, I will point out and try to analyze why this parameter needs to be modified in practical applications.

- Trim_top_db

In the paper hparam, trim_top_db = 40 and it resets to 45 in the implementation.

In the context of audio denoising, "top_db" refers to the threshold of volume in decibels (dB)

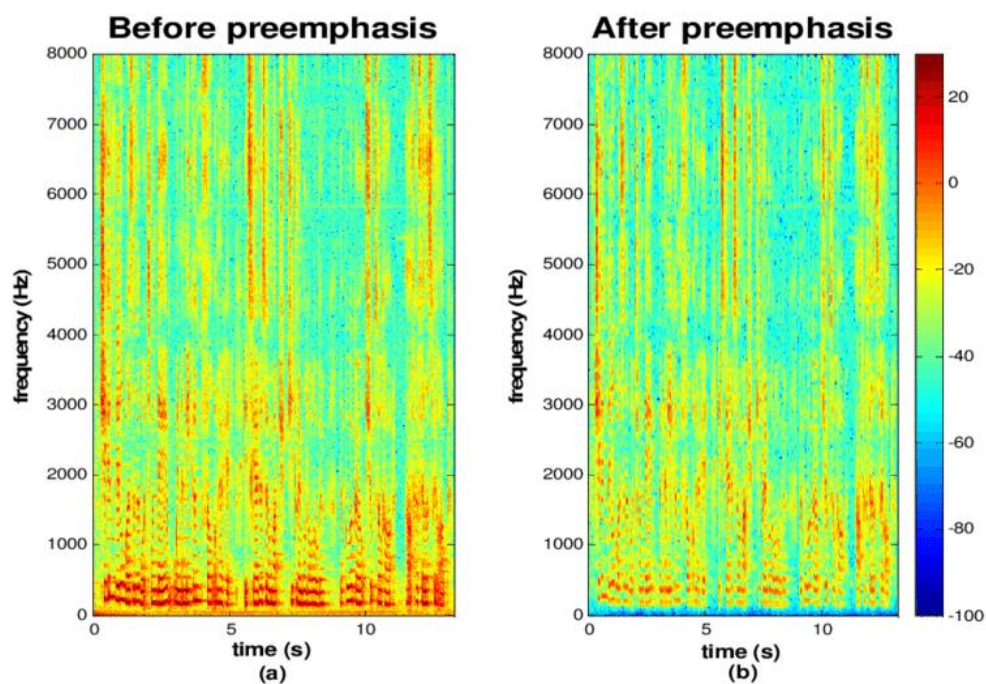
above the noise floor, below which audio signals are considered to be part of the noise and potentially removed or trimmed from the audio signal.

As the "top_db" parameter is increased, fewer audio signals are considered to be part of the noise and therefore are less likely to be removed or trimmed. This can result in a less aggressive denoising process, which may be desirable in some cases to help preserve the quality of the audio signal or avoid cutting off important audio information. And I think this is the reason why they want to increase the number, to keep more audio information in practice.

- Pre-Emphasis

In the paper, Spectrogram Pre-Emphasis = false, but in implementation it sets to true. Spectrogram preemphasis is often used in speech processing applications to remove high-frequency noise from audio signals before computing their spectrograms or other frequency-domain representations. The preemphasis filter amplifies the high-frequency components of the signal, which can help to improve the signal-to-noise ratio and make it easier to analyze speech signals.

For this parameter, I don't think the paper intends to not use it, but to set false so they leave the user one more option to choose. I guess it only sets it false, for someone who wants to use it, they can just set it as true, it seems like a function provided that you can decide whether to use it.



- Fmin

It is a param to set the speaker's gender, and the difference is only depends on different situation, so I guess this is not an important change.

- GL_on_GPU

It is an addition for the paper version, in paper version there is no such a param but it is added in the implementation one. It decides whether to use G&L GPU version as part of tensorflow graph. (Usually much faster than CPU but slightly worse quality too). The real

implement sets it true, I assume for higher speed, since if you want higher quality, you can just use wavenet instead of GL.

- **Clip_outputs**

In the implementation, it added the clip_outputs=true in tacotron model general type, I guess it is a technique used to ensure that the model produces predictions that fall within a desired range and to prevent it from learning to produce out-of-range values.

- **lower_bound_decay**

Also in the implementation, it added lower_bound_decay = true, and it has a explain as Small regularizer for noise synthesis by adding small range of penalty for silence regions.

- **predict_linear**

In the paper, it sets the predict_linear as false, while in the real one it changes it to true. The difference is, if you set this param to true, you can generate the linear spectrograms during synthesizing along with the mel spectrograms. I still don't know why they will set the param to true, because I could not find the function of the linear spectrograms in tacotron-2, I guess they just want to check that or test that?

- **Legacy**

In paper legacy = false but in real it sets to true.

I don't fully understand the function inside the legacy mode, but I think it is another mode that Wavenet can work? And it is suitable for a large model, this is exactly our situation, so it sets it as true to increase the model's training stability.

- **Model parameters**

It changes the parameters to

out_channels = 2, #This should be equal to quantize channels when input type is 'mulaw-quantize'

layers = 20, #Number of dilated convolutions (Default: Simplified Wavenet of Tacotron-2 paper)

stacks = 2, #Number of dilated convolution stacks (Default: Simplified Wavenet of Tacotron-2 paper)

residual_channels = 128, #Number of residual block input/output channels.

gate_channels = 256, #split in 2 in gated convolutions

skip_out_channels = 128, #Number of residual block skip convolution channels.

kernel_size = 3, #The number of inputs to consider in dilated convolutions.

Overall, the wavenet size is simplified when implementing, I guess the wavenet is slow itself when synthesizing, to simplify the model can increase the speed and be more practical.

- **Upsample_conditional_features**

In paper, it sets this param to true, but in implementing, it deletes this param, I search for this param and it said that, if upsample_conditional_features is set to True, the conditional features are upsampled to match the same time resolution as the mel-scale spectrograms. This can help improve the accuracy of the model, since the decoder can more effectively make use of the additional information provided by the conditional features. However, it also increases the computational complexity of the model.

If upsample_conditional_features is set to False, the conditional features are not upsampled and are instead used at their original time resolution. This can reduce the

computational complexity of the model, but may lead to some loss of accuracy since the decoder may have a harder time making use of the additional information provided by the conditional features.

So to delete it is to trade the complexity with loss of accuracy I guess, also suitable for a large model.

- Upsample type

In paper the upsampling deconvolution type is 2D while in implementation is subpixel.

Subpixel convolution is a technique that is specifically designed for upscaling the spatial resolution of an image, and often works better than conventional 2D convolution when the scaling factor is large. So to use subpixel instead of 2D could be often the best choice for generating high-quality speech.

- wavenet_learning_rate

In paper the `wavenet_learning_rate` = $1e-4$, while in the practice, it resets to `wavenet_learning_rate` = $1e-3$.

In general, the learning rate controls how quickly the model parameters are updated during training. A smaller learning rate usually means slower training, but may result in a more accurate and stable model. The optimal value for the learning rate depends on the specific model architecture, dataset, and training task.

So, to change it to a bigger number, the model may decrease in accuracy, but faster training, making the wavenet more practical in generating the waveform.

- tacotron_reg_weight

It is a parameter for regularization weight, in paper it is $1e-7$, while in practice is $1e-6$. A larger regularization weight typically means that the model is penalized more for having large weights, which can help prevent overfitting and improve the model's generalization performance on unseen data. So I guess it can improve the general using of the model

- Conclusion

I feel really out of energy now, I still have many difference to compare, but my mind just go empty now, so this could be the first version of the report, I will submit it as the assignment but I will explore more on the hparams which can tell me lots of information on the details of tacotron2.

After comparing the hparams in the paper and in the real implementation, I think generally it has two tendency.

Firstly is the tendence from paper to practice is simplifying the structure of the model and speeding up model training, especially for the Wavenet. Since in practice, especially for a large model like tacotron2, it may need more speed than accuracy and quality, and the wavenet already have a good quality so it may need more speed. In the paper, it definitely will pursue a ideal model, but it sometimes wastes time in practice. Especially for tacotron2, which is already slower than fastspeech.

The second tendence I guess is the more general use ability. Since the model need to be more flexible in dealing with different tasks, with different languages. So in practical use, it also needs to trade general use with accuracy.