Training BERT-VITS2: Voice Synthesis of Virtual Characters

Yaoen Shi
ys5560@nyu.edu

Abstract

In this project, I tried to train a multi-speaker Bert-VITS2 model that can convert text into speech for Chinese language. The code is based on a github repository Bert-VITS2 and the data is downloaded from another repository Genshin-Datasets. The training spent around 17 hours on my NVIDIA GeForce RTX 3080 Ti GPU and the result is pushed to my github account. Please refer to the README.md in that repository to help you check the quality of the final model: `https://github.com/sheyehs/Bert-VITS2-training#readme`.

Introduction

TTS(text-to-speech) is a task in the field of natural language processing. It aims to generate realistic and natural human speech when the text content is provided. In the sense of data type conversion, it transforms text files into audio files like .wav format. For a long time, the two-stage TTS models had been prevalent in research domain, for example, Glow-TTS, FastSpeech and Tacotron series. They generally consist of a acoustic model that generates acoustic features from linguistic features and a vocoder that synthesizes a waveform from those acoustic features. Along with the development in Large Language Models, another type of one-stage TTS models emerged in recent years, among which the representative is VITS (Kim et al., 2021(0)). It absorbs new techniques and architectures from LLM such as Word Embedding, Transformer, Variational Auto-Encoder, GAN, Flow and so on. It is lightweight, trained end-to-end and can achieve better performance in the meantime. Benefited from VITS and its subsequent modification VITS2 (Kong et al., 2023(0)), individuals now are able to fine-tune TTS models on personal computers as long as a pre-trained base model is available. What us ordinary people can do with TTS models? Some enthusiasts have successfully explored it on the voice synthesis of virtual characters, such as from games, anime or visual novels, which are cast by real human actors, and they can reproduce the sound magically before the computer screen or even build a chat robot using these virtual sounds. Inspired by an excellent video example, I decided to try to train this complicated and large (for individuals) model on my own computer.

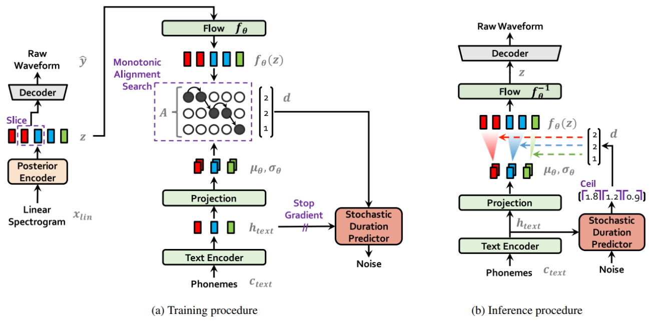


Figure 1. System diagram depicting (a) training procedure and (b) inference procedure. The proposed model can be viewed as a conditional VAE; a posterior encoder, decoder, and conditional prior (green blocks: a normalizing flow, linear projection layer, and text encoder) with a flow-based stochastic duration predictor.

Figure 1: The model architecture in the paper of VITS

Literature Survey

1. VITS

Inference The full name of VITS is Variational Inference with adversarial learning for end-to-end Text-to-Speech. During inference, it consists of following steps:

- A prior encoder to extract features from tokenized text. It process the input that we want the TTS model to speak.
- A stochastic duration predictor to predict the duration of each text token.
- One word, i.e., one text token, always corresponds to multiple voice elements, named phonemes, so an easy alignment exists here to do this mapping.
- A Flow network transforms the aligned features (now each one corresponds to a phoneme, not a word) to the latent space of speech. Now we have the truly voice features.
- Finally, a speech decoder converts the speech features into waveforms, as the output we can listen to.

Training While in training, the situation becomes a little bit complicated and subtle. It combines a variational auto-encoder for the phonemes and a adversarial training process, as the full name of VITS suggests. Therefore, it has three more components now:

- Since the stochastic duration predictor is not ready yet and we know the ground-truth correspondence from phonemes to words. But we do not know the time duraion of each phoneme, so a dynamic programming algorithm, called monotonic alignment search, is added to complete the alignment task.
- A posteprior encoder which takes into the ground-truth phonemes works with the decoder in order to improve the representation of the voice latent space. They construct not only a vanilla auto-encoder, but a variational auto-encoder that predicts the mean and variance of the latent features, thus adding more randomness and diversity to the generated voice.
- A GAN discriminator judges wether it is a real or fake voice at the very last stage to help enhance the synthesis quality.

Note that the voice waveform data, no matter in training or inference modes, is actually represented by Mel-Spectrogram, an acoustic feature that cannot be listened directly but is able to represent the voice better. It can be converted from or to waveform using mathematical formulas.

Losses The model is trained end-to-end with a combination of losses derived from variational lower bound and adversarial training. Some losses are explicit to understand, such as the reconstruction loss in the auto-encoder, it is simply the difference of the predictions and ground-truths:

$$L_{recon} = \|x_{mel} - \hat{x}_{mel}\|_1$$

A KL divergence loss is used to measure if the distribution of latent variables from phonemes matches with the distribution of latent variable from words:

$$L_{kl} = \log q_{\phi}(z|x_{lin}) - \log p_{\theta}(z|c_{text}, A)$$

Of course with the loss from the stochastic duration predictor (it is internally another small model) and the loss from GAN training:

$$\begin{aligned}
L_{adv}(D) &= \mathbb{E}_{(y,z)}\Big[(D(y)-1)^2 + (D(G(z)))^2\Big], \\
L_{adv}(G) &= \mathbb{E}_{z}\Big[(D(G(z))-1)^2\Big], \\
L_{fm}(G) &= \mathbb{E}_{(y,z)}\Big[\sum_{l=1}^{T} \frac{1}{N_l}\|D^l(y) - D^l(G(z))\|_1\Big]
\end{aligned}$$

The first two GAN losses are what we are familiar with and taught on the course. And the third one is an additional loss to compare in the latent space inside the discriminator in the sense of that the synthesized voice comes from the real voice, so the generator is trying to make the similar features. Finally, all features are summed up together as the total loss

$$L_{vae} = L_{recon} + L_{kl} + L_{dur} + L_{adv}(G) + L_{fm}$$

and remember that the discriminator loss is updated alternatively in another gradient backward:

2. VITS2

VITS2 actually does not change the overall model architecture, but improve some stages internally with new discoveries and techniques:

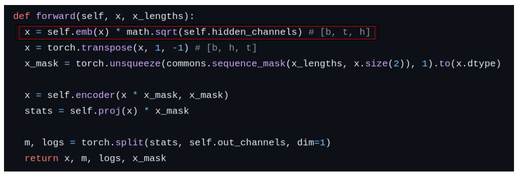
- Most significantly, it replaces the 1D convolution blocks with the fancier Transformer blocks.
- Add adversarial learning to the stochastic durion predictor. Now we have two GAN training processes.
- Add Gaussian noise to Monotonic Alignment Search to make the alignment more diverse because we human often vary our speeding duration even for the same word.
- Not only voice encoder has the speaker condition, but also add the condition to text encoder.

Technical Details

Above is the introduction of the original VITS and VITS2 models. Bert-VIST2 is a modification version of VITS2 by community enthusiasts. Bert (Devlin et al., 2019(0)) can provide mature word embedding that can be directly utilized. Community enthusiasts put the Bert word embedding at the very beginning stage of the priori encoder so the model can take into these good existing features and thus accelerates the training. Here we describe the new changes about Bert-VITS2 and the configuration details of my training process.

1. Bert Embedding

We can found that Bert embedding is truly added in the forward process of the TextEncoder:

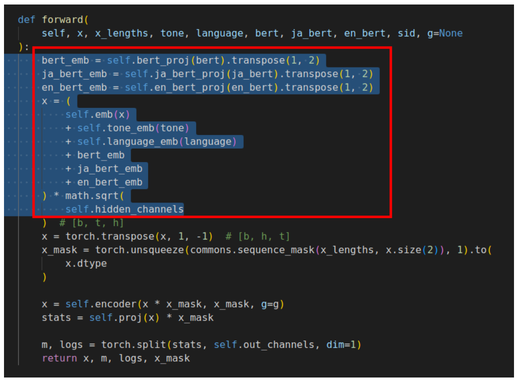


```
def forward(self, x, x_lengths):
  x = self.emb(x) * math.sqrt(self.hidden_channels) # [b, t, h]
  x = torch.transpose(x, 1, -1) # [b, h, t]
  x_mask = torch.unsqueeze(commons.sequence_mask(x_lengths, x.size(2)), 1).to(x.dtype)

  x = self.encoder(x * x_mask, x_mask)
  stats = self.proj(x) * x_mask

  m, logs = torch.split(stats, self.out_channels, dim=1)
  return x, m, logs, x_mask
```

Figure 2: The original implementation simply uses the embedding which is obtained during the training: `https://github.com/jaywalnut310/vits/blob/2e561ba58618d021b5b8323d3765880f7e0ecfdb/models.py#L168C12-L168C12`



```python
def forward(
    self, x, x_lengths, tone, language, bert, ja_bert, en_bert, sid, g=None
):
    bert_emb = self.bert_proj(bert).transpose(1, 2)
    ja_bert_emb = self.ja_bert_proj(ja_bert).transpose(1, 2)
    en_bert_emb = self.en_bert_proj(en_bert).transpose(1, 2)
    x = (
        self.emb(x)
        + self.tone_emb(tone)
        + self.language_emb(language)
        + bert_emb
        + ja_bert_emb
        + en_bert_emb
    ) * math.sqrt(
        self.hidden_channels
    )  # [b, t, h]
    x = torch.transpose(x, 1, -1)  # [b, h, t]
    x_mask = torch.unsqueeze(commons.sequence_mask(x_lengths, x.size(2)), 1).to(
        x.dtype
    )

    x = self.encoder(x * x_mask, x_mask, g=g)
    stats = self.proj(x) * x_mask

    m, logs = torch.split(stats, self.out_channels, dim=1)
    return x, m, logs, x_mask
```

Figure 3: Bert-VITS2 appends Bert embedding to the original embedding

2. Configuration Changes

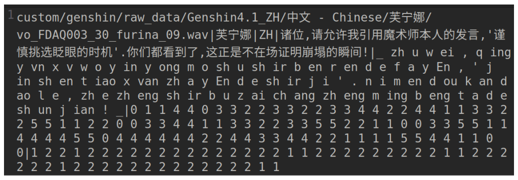
Since VITS(2) is a multi-speaker TTS model, which guides the generation with the speaker index as condition, I examine this mechanism by training on two virtual characters in the video game Genshin Impact, named Furina and Hutao.

On the first several attempts, I found my computer is still limited in capacity and the error: "CUDA is out of memory" always occurred. Then I figured out to reduce the batch size from default 12 to 8 and fortunately solved this problem.

I left other hyper-parameters unchanged (because there are too many to tune!) and the results seemed like not bad with these default settings, showing the robustness of the modern large neural networks.

3. Dataset Preparation

But I still need to prepare my own dataset. Luckily, there exists a repository as mentioned at the very beginning of this report, which provides the necessary data for TTS training. The data required by the model should be records that consist of plain text, speaking character name and audio file path. Also, the raw format from the dataset does not match the rules specified by the model, so I wrote a Python script to complete the format conversion. And, please notice, the model actually does not take into words that we read everyday! It only takes into phonemes that marks how people pronounce them but not how people write down them. The good news is that every language has a rule to map the text to its prononciation, although in different description of sounds, but keep consistent inside one language. Some people already create the script to do this task. At the end stage of the data preparation, I ran that script and obtained the Pinyin (a system using Latin letters and four numbers) for Chinese language. This is the example of the final thing I got:



```
custom/genshin/raw_data/Genshin4.1_ZH/中文 - Chinese/芙宁娜/
vo_FDAQ003_30_furina_09.wav|芙宁娜|ZH|诸位,请允许我引用魔术师本人的发言,'谨
慎挑选眨眼的时机'.你们都看到了,这正是不在场证明崩塌的瞬间!|_ zh u w ei , q ing
y vn x v w o y in y ong m o sh u sh ir b en r en d e f a y En , ' j
in sh en t iao x van zh a y En d e sh ir j i ' . n i m en d ou k an d
ao l e , zh e zh eng sh ir b u z ai ch ang zh eng m ing b eng t a d e
sh un j ian ! _|0 1 1 4 4 0 3 3 2 2 3 3 2 2 3 3 4 4 2 2 4 4 1 1 3 3 2
2 5 5 1 1 2 2 0 0 3 3 4 4 1 1 3 3 2 2 3 3 5 5 2 2 1 1 0 0 3 3 5 5 1 1
4 4 4 4 5 5 0 4 4 4 4 4 4 2 2 4 4 3 3 4 4 2 2 1 1 1 1 5 5 4 4 1 1 0
0|1 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 2 2 2 2 2 2 2 2 2 1 1 2 2 2
2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1
```

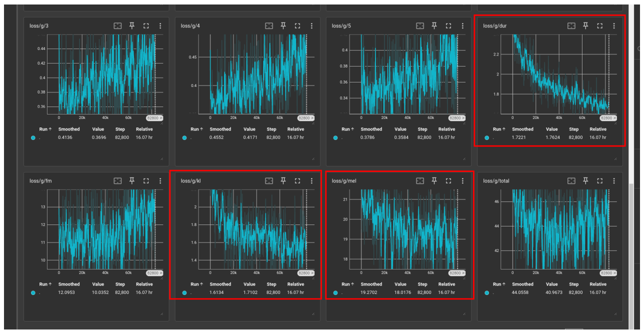
Figure 4: Example of a training sequence: by order you can find the audio file path, character name, language type, Chinese text, Chinese Pinyin and speech duration

Training Results

There are 695 training sequences in total, 125 for Funina and 570 for Hutao. Two characters are trained together and distinguished by the index condition. The batch size is 8 and the number of epochs is 1000, thus training $695 * 1000/8 = \approx 86875$ iterations. The final training spent around 17 hours on my personal PC. The training is fine-tuning, based on a pre-trained model which has the voice of a standard male broadcaster, and the final models has the voice of two game female characters.

1. Tensorboard Metrics

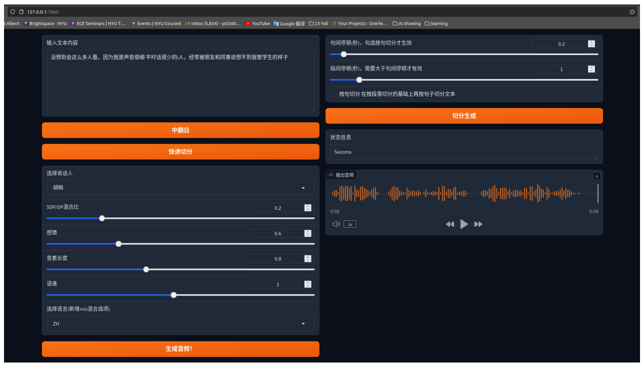
All metrics are recorded in a tensorboard event file and there is a GUI webpage to visualize them:



The loss of stochastic predictor, kl divergence and mel-spectrogram (reconstruction) was been decreasing, which showed that the training was converging. Do not be fooled by the increasing of generator or discriminator loss because they were competing with each other and the value of their losses were affected by their relative ability.

2. Generation from New Sentences

While for training launching the process in command lines, for inference the original repository provides a web GUI interface based on Gradio Python library to make it convenient to input text sentences and play the generated audio. If you have played with the popular stable-diffusion Web GUI, I believe you must be familiar with this kind of webpage: In the above screenshot, I tried the sentence which translated into English was "I didn't expect that so many people would see it, because I have a very thin voice and rarely talk. My friends and colleagues often say that I don't think I care about my students" and got a great result.



输入文本内容
没想到会这么多人看，因为我是声音很细 平时话很少的i人，经常被朋友和同事说想不到我管学生的样子

选择说话人: 胡桃
SDP/DP混合比: 0.2
感情: 0.6
音素长度: 0.8
语速: 1
选择语言(新增mix混合选项): ZH

句间停顿(秒)，勾选按句切分才生效: 0.2
段间停顿(秒)，需要大于句间停顿才有效: 1
☐ 按句切分 在按段落切分的基础上再按句子切分文本

状态信息
Success

3. How You Examine It

Wait, you might say, ”I am an English speaker and how do I know if you train the model successfully and generate good voices?”. Well, I think of a way to deal with it. Please follow the instructions in the README.md at my repository. That file includes how to prepare the environment and how to examine the synthesis quality for English graders.

Conclusion

In this project, I trained a language model. What I wanted to achieve when I prepared the proposal is to make the virtual characters look like alive by speaking arbitrary sentences in their distinctive voices. Actually, before choosing this proposal, my initial plan is to create a virtual chat robot floating on the desktop screen. It can recognize my sound as text input, return the response using ChatGPT, speak out the response and produce semantic feelings. All of these functionalities are accomplished by neural networks and have a basic framework that can be ran right now. Next step, I can replace the remote Text-to-Speech module with the one I trained here, thus saving the time of transferring data along the Internet and making the whole system more stable.

References

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

Jaehyeon Kim, Jungil Kong, and Juhee Son. Conditional variational autoencoder with adversarial learning for end-to-end text-to-speech, 2021.

Jungil Kong, Jihoon Park, Beomjeong Kim, Jeongmin Kim, Dohee Kong, and Sangjin Kim. Vits2: Improving quality and efficiency of single-stage text-to-speech with adversarial learning and architecture design, 2023.