TTS

Search

Fine-tuning a 🐸 TTS model

Fine-tuning

Fine-tuning takes a pre-trained model and retrains it to improve the model performance on a different task or dataset. In 🐸TTS we provide different pre-trained models in different languages and different pros and cons. You can take one of them and fine-tune it for your own dataset. This will help you in two main ways:

1. Faster learning

   Since a pre-trained model has already learned features that are relevant for the task, it will converge faster on a new dataset. This will reduce the cost of training and let you experiment faster.

2. Better results with small datasets

   Deep learning models are data hungry and they give better performance with more data. However, it is not always possible to have this abundance, especially in specific domains. For instance, the LJSpeech dataset, that we released most of our English models with, is almost 24 hours long. It takes weeks to record this amount of data with the help of a voice actor.

   Fine-tuning comes to the rescue in this case. You can take one of our pre-trained models and fine-tune it on your own speech dataset and achieve reasonable results with only a couple of hours of data.

   However, note that, fine-tuning does not ensure great results. The model performance still depends on the dataset quality and the hyper-parameters you choose for fine-tuning. Therefore, it still takes a bit of tinkering.

Steps to fine-tune a 🐸 TTS model

1. Setup your dataset.

   You need to format your target dataset in a certain way so that 🐸TTS data loader will be able to load it for the training. Please see this page for more information about formatting.

2. Choose the model you want to fine-tune.

   You can list the available models in the command line with

   ```
   tts --list_models
   ```

   The command above lists the models in a naming format as `<model_type>/<language>/<dataset>/<model_name>`.

   Or you can manually check the `.model.json` file in the project directory.

   You should choose the model based on your requirements. Some models are fast and some are better in speech quality. One lazy way to test a model is running the model on the hardware you want to use and see how it works. For simple testing, you can use the `tts` command on the terminal. For more info see here.

3. Download the model.

   You can download the model by using the `tts` command. If you run `tts` with a particular model, it will download it automatically and the model path will be printed on the terminal.

   ```
   tts --model_name tts_models/es/mai/tacotron2-DDC --text "Ola."

   > Downloading model to /home/ubuntu/.local/share/tts/tts_models--en--ljspeech--glow-tts
   ...
   ```

   In the example above, we called the Spanish Tacotron model and give the sample output showing use the path where the model is downloaded.

4. Setup the model config for fine-tuning.

   You need to change certain fields in the model config. You have 3 options for playing with the configuration.

   1. Edit the fields in the `config.json` file if you want to use `TTS/bin/train_tts.py` to train the model.
   2. Edit the fields in one of the training scripts in the `recipes` directory if you want to use python.
   3. Use the command-line arguments to override the fields like `--coqpit.lr 0.00001` to change the learning rate.

   Some of the important fields are as follows:

   - `datasets` field: This is set to the dataset you want to fine-tune the model on.
   - `run_name` field: This is the name of the run. This is used to name the output directory and the entry in the logging dashboard.
   - `output_path` field: This is the path where the fine-tuned model is saved.
   - `lr` field: You may need to use a smaller learning rate for fine-tuning to not lose the features learned by the pre-trained model with big update steps.
   - `audio` fields: Different datasets have different audio characteristics. You must check the current audio parameters and make sure that the values reflect your dataset. For instance, your dataset might have a different audio sampling rate.

   Apart from the parameters above, you should check the whole configuration file and make sure that the values are correct for your dataset and training.

5. Start fine-tuning.

   Whether you use one of the training scripts under `recipes` folder or the `train_tts.py` to start your training, you should use the `--restore_path` flag to specify the path to the pre-trained model.

   ```
   CUDA_VISIBLE_DEVICES="0" python recipes/ljspeech/glow_tts/train_glowtts.py \
       --restore_path  /home/ubuntu/.local/share/tts/tts_models--en--ljspeech--glow-tts/model
   ```

   ```
   CUDA_VISIBLE_DEVICES="0" python TTS/bin/train_tts.py \
       --config_path  /home/ubuntu/.local/share/tts/tts_models--en--ljspeech--glow-tts/config
       --restore_path  /home/ubuntu/.local/share/tts/tts_models--en--ljspeech--glow-tts/model
   ```

   As stated above, you can also use command-line arguments to change the model configuration.

   ```
   CUDA_VISIBLE_DEVICES="0" python recipes/ljspeech/glow_tts/train_glowtts.py \
       --restore_path  /home/ubuntu/.local/share/tts/tts_models--en--ljspeech--glow-tts/model
       --coqpit.run_name "glow-tts-finetune" \
       --coqpit.lr 0.00001
   ```