
Status: Archive (code is provided as-is, no updates expected)

Jukebox

Code for “Jukebox: A Generative Model for Music”

Paper Blog Explorer Colab

Install

Install the conda package manager from <https://docs.conda.io/en/latest/miniconda.html>

```
1 # Required: Sampling
2 conda create --name jukebox python=3.7.5
3 conda activate jukebox
4 conda install mpi4py=3.0.3 # if this fails, try: pip install mpi4py
   ==3.0.3
5 conda install pytorch=1.4 torchvision=0.5 cudatoolkit=10.0 -c pytorch
6 git clone https://github.com/openai/jukebox.git
7 cd jukebox
8 pip install -r requirements.txt
9 pip install -e .
10
11 # Required: Training
12 conda install av=7.0.01 -c conda-forge
13 pip install ./tensorboardX
14
15 # Optional: Apex for faster training with fused_adam
16 conda install pytorch=1.1 torchvision=0.3 cudatoolkit=10.0 -c pytorch
17 pip install -v --no-cache-dir --global-option="--cpp_ext" --global-
   option="--cuda_ext" ./apex
```

Sampling

Sampling from scratch

To sample normally, run the following command. Model can be 5b, 5b_lyrics, 1b_lyrics

```
1 python jukebox/sample.py --model=5b_lyrics --name=sample_5b --levels=3
   --sample_length_in_seconds=20 \
2 --total_sample_length_in_seconds=180 --sr=44100 --n_samples=6 --
   hop_fraction=0.5,0.5,0.125
```

```
1 python jukebox/sample.py --model=1b_lyrics --name=sample_1b --levels=3
  --sample_length_in_seconds=20 \
2 --total_sample_length_in_seconds=180 --sr=44100 --n_samples=16 --
  hop_fraction=0.5,0.5,0.125
```

The above generates the first `sample_length_in_seconds` seconds of audio from a song of total length `total_sample_length_in_seconds`. To use multiple GPU's, launch the above scripts as `mpiexec -n {ngpus} python jukebox/sample.py ...` so they use `{ngpus}`

The samples decoded from each level are stored in `{name}/level_{level}`. You can also view the samples as an html with the aligned lyrics under `{name}/level_{level}/index.html`. Run `python -m http.server` and open the html through the server to see the lyrics animate as the song plays.

A summary of all sampling data including zs, x, labels and sampling_kwargs is stored in `{name}/level_{level}/data.pth.tar`.

The hps are for a V100 GPU with 16 GB GPU memory. The `1b_lyrics`, `5b`, and `5b_lyrics` top-level priors take up 3.8 GB, 10.3 GB, and 11.5 GB, respectively. The peak memory usage to store transformer key, value cache is about 400 MB for `1b_lyrics` and 1 GB for `5b_lyrics` per sample. If you are having trouble with CUDA OOM issues, try `1b_lyrics` or decrease `max_batch_size` in `sample.py`, and `--n_samples` in the script call.

On a V100, it takes about 3 hrs to fully sample 20 seconds of music. Since this is a long time, it is recommended to use `n_samples > 1` so you can generate as many samples as possible in parallel. The 1B lyrics and upsamplers can process 16 samples at a time, while 5B can fit only up to 3. Since the vast majority of time is spent on upsampling, we recommend using a multiple of 3 less than 16 like `--n_samples 15` for `5b_lyrics`. This will make the top-level generate samples in groups of three while upsampling is done in one pass.

To continue sampling from already generated codes for a longer duration, you can run

```
1 python jukebox/sample.py --model=5b_lyrics --name=sample_5b --levels=3
  --mode=continue \
2 --codes_file=sample_5b/level_0/data.pth.tar --sample_length_in_seconds
  =40 --total_sample_length_in_seconds=180 \
3 --sr=44100 --n_samples=6 --hop_fraction=0.5,0.5,0.125
```

Here, we take the 20 seconds samples saved from the first sampling run at `sample_5b/level_0/data.pth.tar` and continue by adding 20 more seconds.

You could also continue directly from the level 2 saved outputs, just pass `--codes_file=sample_5b/level_2/data.pth.tar`. Note this will upsample the full 40 seconds song at the end.

If you stopped sampling at only the first level and want to upsample the saved codes, you can run

```
1 python jukebox/sample.py --model=5b_lyrics --name=sample_5b --levels=3
  --mode=upsample \
2 --codes_file=sample_5b/level_2/data.pth.tar --sample_length_in_seconds
  =20 --total_sample_length_in_seconds=180 \
3 --sr=44100 --n_samples=6 --hop_fraction=0.5,0.5,0.125
```

Here, we take the 20 seconds samples saved from the first sampling run at `sample_5b/level_2/data.pth.tar` and upsample the lower two levels.

Prompt with your own music

If you want to prompt the model with your own creative piece or any other music, first save them as wave files and run

```
1 python jukebox/sample.py --model=5b_lyrics --name=sample_5b_prompted --
  levels=3 --mode=primed \
2 --audio_file=path/to/recording.wav,awesome-mix.wav,fav-song.wav,etc.wav
  --prompt_length_in_seconds=12 \
3 --sample_length_in_seconds=20 --total_sample_length_in_seconds=180 --sr
  =44100 --n_samples=6 --hop_fraction=0.5,0.5,0.125
```

This will load the four files, tile them to fill up to `n_samples` batch size, and prime the model with the first `prompt_length_in_seconds` seconds.

Training

VQVAE

To train a small vqvae, run

```
1 mpiexec -n {ngpus} python jukebox/train.py --hps=small_vqvae --name=
  small_vqvae --sample_length=262144 --bs=4 \
2 --audio_files_dir={audio_files_dir} --labels=False --train --aug_shift
  --aug_blend
```

Here, `{audio_files_dir}` is the directory in which you can put the audio files for your dataset, and `{ngpus}` is number of GPU's you want to use to train. The above trains a two-level VQ-VAE with `downs_t = (5, 3)`, and `strides_t = (2, 2)` meaning we downsample the audio by $2 \times 5 = 32$ to get the first level of codes, and $2 \times 8 = 256$ to get the second level codes.

Checkpoints are stored in the `logs` folder. You can monitor the training by running Tensorboard

```
1 tensorboard --logdir logs
```

Prior

Train prior or upsamplers

Once the VQ-VAE is trained, we can restore it from its saved checkpoint and train priors on the learnt codes. To train the top-level prior, we can run

```
1 mpiexec -n {ngpus} python jukebox/train.py --hps=small_vqvae,
  small_prior,all_fp16,cpu_ema --name=small_prior \
2 --sample_length=2097152 --bs=4 --audio_files_dir={audio_files_dir} --
  labels=False --train --test --aug_shift --aug_blend \
3 --restore_vqvae=logs/small_vqvae/checkpoint_latest.pth.tar --prior --
  levels=2 --level=1 --weight_decay=0.01 --save_iters=1000
```

To train the upsampler, we can run

```
1 mpiexec -n {ngpus} python jukebox/train.py --hps=small_vqvae,
  small_upsampler,all_fp16,cpu_ema --name=small_upsampler \
2 --sample_length=262144 --bs=4 --audio_files_dir={audio_files_dir} --
  labels=False --train --test --aug_shift --aug_blend \
3 --restore_vqvae=logs/small_vqvae/checkpoint_latest.pth.tar --prior --
  levels=2 --level=0 --weight_decay=0.01 --save_iters=1000
```

We pass `sample_length = n_ctx * downsample_of_level` so that after downsampling the tokens match the `n_ctx` of the prior hps. Here, `n_ctx = 8192` and `downsamples = (32, 256)`, giving `sample_lengths = (8192 * 32, 8192 * 256) = (65536, 2097152)` respectively for the bottom and top level.

Learning rate annealing

To get the best sample quality anneal the learning rate to 0 near the end of training. To do so, continue training from the latest checkpoint and run with

```
1 --restore_prior="path/to/checkpoint" --lr_use_linear_decay --
  lr_start_linear_decay={already_trained_steps} --lr_decay={
  decay_steps_as_needed}
```

Reuse pre-trained VQ-VAE and train top-level prior on new dataset from scratch.

Train without labels Our pre-trained VQ-VAE can produce compressed codes for a wide variety of genres of music, and the pre-trained upsamplers can upsample them back to audio that sound very similar to the original audio. To re-use these for a new dataset of your choice, you can retrain just the top-level

To train top-level on a new dataset, run

```
1 mpiexec -n {ngpus} python jukebox/train.py --hps=vqvae,small_prior,
  all_fp16,cpu_ema --name=pretrained_vqvae_small_prior \
2 --sample_length=1048576 --bs=4 --aug_shift --aug_blend --
  audio_files_dir={audio_files_dir} \
3 --labels=False --train --test --prior --levels=3 --level=2 --
  weight_decay=0.01 --save_iters=1000
```

Training the `small_prior` with a batch size of 2, 4, and 8 requires 6.7 GB, 9.3 GB, and 15.8 GB of GPU memory, respectively. A few days to a week of training typically yields reasonable samples when the dataset is homogeneous (e.g. all piano pieces, songs of the same style, etc).

Near the end of training, follow this to anneal the learning rate to 0

Sample from new model You can then run `sample.py` with the top-level of our models replaced by your new model. To do so, - Add an entry `my_model= ("vqvae", "upsampler_level_0", "upsampler_level_1", "small_prior")` in `MODELS` in `make_models.py`. - Update the `small_prior` dictionary in `hparams.py` to include `restore_prior='path/to/checkpoint'`. If you changed any hps directly in the command line script (eg: `heads`), make sure to update them in the dictionary too so that `make_models` restores our checkpoint correctly. - Run `sample.py` as outlined in the sampling section, but now with `--model=my_model`

For example, let's say we trained `small_vqvae`, `small_prior`, and `small_upsampler` under `/path/to/jukebox/logs`. In `make_models.py`, we are going to declare a tuple of the new models as `my_model`.

```
1 MODELS = {
2     '5b': ("vqvae", "upsampler_level_0", "upsampler_level_1", "prior_5b"),
3     '5b_lyrics': ("vqvae", "upsampler_level_0", "upsampler_level_1", "prior_5b_lyrics"),
4     '1b_lyrics': ("vqvae", "upsampler_level_0", "upsampler_level_1", "prior_1b_lyrics"),
5     'my_model': ("my_small_vqvae", "my_small_upsampler", "my_small_prior"),
6 }
```

Next, in `hparams.py`, we add them to the registry with the corresponding `restore_paths` and any other command line options used during training. Another important note is that for top-level priors with lyric conditioning, we have to locate a self-attention layer that shows alignment between the lyric and music tokens. Look for layers where `prior.prior.transformer._attn_mods[layer].attn_func` is either 6 or 7. If your model is starting to sing along lyrics, it means some layer, head pair has learned alignment. Congrats!

```

1 my_small_vqvae = Hyperparams(
2     restore_vqvae='/path/to/jukebox/logs/small_vqvae/
      checkpoint_some_step.pth.tar',
3 )
4 my_small_vqvae.update(small_vqvae)
5 HPARAMS_REGISTRY["my_small_vqvae"] = my_small_vqvae
6
7 my_small_prior = Hyperparams(
8     restore_prior='/path/to/jukebox/logs/small_prior/checkpoint_latest.
      pth.tar',
9     level=1,
10    labels=False,
11    # TODO For the two lines below, if `--labels` was used and the
      model is
12    # trained with lyrics, find and enter the layer, head pair that has
      learned
13    # alignment.
14    alignment_layer=47,
15    alignment_head=0,
16 )
17 my_small_prior.update(small_prior)
18 HPARAMS_REGISTRY["my_small_prior"] = my_small_prior
19
20 my_small_upsampler = Hyperparams(
21     restore_prior='/path/to/jukebox/logs/small_upsampler/
      checkpoint_latest.pth.tar',
22     level=0,
23     labels=False,
24 )
25 my_small_upsampler.update(small_upsampler)
26 HPARAMS_REGISTRY["my_small_upsampler"] = my_small_upsampler

```

Train with labels To train with your own metadata for your audio files, implement `get_metadata` in `data/files_dataset.py` to return the `artist`, `genre` and `lyrics` for a given audio file. For now, you can pass `' '` for lyrics to not use any lyrics.

For training with labels, we'll use `small_labelled_prior` in `hparams.py`, and we set `labels=True, labels_v3=True`. We use 2 kinds of labels information: - Artist/Genre: - For each file, we return an `artist_id` and a list of `genre_ids`. The reason we have a list and not a single `genre_id` is that in v2, we split genres like `blues_rock` into a bag of words [`blues`, `rock`], and we pass at most `max_bow_genre_size` of those, in v3 we consider it as a single word and just set `max_bow_genre_size=1`. - Update the `v3_artist_ids` and `v3_genre_ids` to use ids from your new dataset. - In `small_labelled_prior`, set the hps `y_bins = (number_of_genres, number_of_artists)` and `max_bow_genre_size=1`. - Timing: - For each chunk of audio, we return the `total_length` of the song, the `offset` the current

audio chunk is at and the `sample_length` of the audio chunk. We have three timing embeddings: `total_length`, our current position, and our current position as a fraction of the total length, and we divide the range of these values into `t_bins` discrete bins. - In `small_labelled_prior`, set the hps `min_duration` and `max_duration` to be the shortest/longest duration of audio files you want for your dataset, and `t_bins` for how many bins you want to discretize timing information into. Note `min_duration * sr` needs to be at least `sample_length` to have an audio chunk in it.

After these modifications, to train a top-level with labels, run

```
1 mpiexec -n {ngpus} python jukebox/train.py --hps=vqvae,
  small_labelled_prior,all_fp16,cpu_ema --name=
  pretrained_vqvae_small_prior_labels \
2 --sample_length=1048576 --bs=4 --aug_shift --aug_blend --
  audio_files_dir={audio_files_dir} \
3 --labels=True --train --test --prior --levels=3 --level=2 --
  weight_decay=0.01 --save_iters=1000
```

For sampling, follow same instructions as above but use `small_labelled_prior` instead of `small_prior`.

Train with lyrics To train in addition with lyrics, update `get_metadata` in `data/files_dataset.py` to return `lyrics` too. For training with lyrics, we'll use `small_single_enc_dec_prior` in `hparams.py`. - Lyrics: - For each file, we linearly align the lyric characters to the audio, find the position in lyric that corresponds to the midpoint of our audio chunk, and pass a window of `n_tokens` lyric characters centred around that. - In `small_single_enc_dec_prior`, set the hps `use_tokens=True` and `n_tokens` to be the number of lyric characters to use for an audio chunk. Set it according to the `sample_length` you're training on so that its large enough that the lyrics for an audio chunk are almost always found inside a window of that size. - If you use a non-English vocabulary, update `text_processor.py` with your new vocab and set `n_vocab = number of characters in vocabulary` accordingly in `small_single_enc_dec_prior`. In v2, we had a `n_vocab=80` and in v3 we missed + and so `n_vocab=79` of characters.

After these modifications, to train a top-level with labels and lyrics, run

```
1 mpiexec -n {ngpus} python jukebox/train.py --hps=vqvae,
  small_single_enc_dec_prior,all_fp16,cpu_ema --name=
  pretrained_vqvae_small_single_enc_dec_prior_labels \
2 --sample_length=786432 --bs=4 --aug_shift --aug_blend --audio_files_dir
  ={audio_files_dir} \
3 --labels=True --train --test --prior --levels=3 --level=2 --
  weight_decay=0.01 --save_iters=1000
```

To simplify hps choices, here we used a `single_enc_dec` model like the `1b_lyrics` model that combines both encoder and decoder of the transformer into a single model. We do so by merging

the lyric vocab and vq-vae vocab into a single larger vocab, and flattening the lyric tokens and the vq-vae codes into a single sequence of length `n_ctx + n_tokens`. This uses `attn_order=12` which includes `prime_attention` layers with keys/values from lyrics and queries from audio. If you instead want to use a model with the usual encoder-decoder style transformer, use `small_sep_enc_dec_prior`.

For sampling, follow same instructions as above but use `small_single_enc_dec_prior` instead of `small_prior`. To also get the alignment between lyrics and samples in the saved html, you'll need to set `alignment_layer` and `alignment_head` in `small_single_enc_dec_prior`. To find which layer/head is best to use, run a forward pass on a training example, save the attention weight tensors for all `prime_attention` layers, and pick the (layer, head) which has the best linear alignment pattern between the lyrics keys and music queries.

Fine-tune pre-trained top-level prior to new style(s)

Previously, we showed how to train a small top-level prior from scratch. Assuming you have a GPU with at least 15 GB of memory and support for fp16, you could fine-tune from our pre-trained 1B top-level prior. Here are the steps:

- Support `--labels=True` by implementing `get_metadata` in `jukebox/data/files_dataset.py` for your dataset.
- Add new entries in `jukebox/data/ids`. We recommend replacing existing mappings (e.g. rename `"unknown"`, etc with styles of your choice). This uses the pre-trained style vectors as initialization and could potentially save some compute.

After these modifications, run

```
1 mpiexec -n {ngpus} python jukebox/train.py --hps=vqvae,prior_1b_lyrics,
   all_fp16,cpu_ema --name=finetuned \
2 --sample_length=1048576 --bs=1 --aug_shift --aug_blend --
   audio_files_dir={audio_files_dir} \
3 --labels=True --train --test --prior --levels=3 --level=2 --
   weight_decay=0.01 --save_iters=1000
```

To get the best sample quality, it is recommended to anneal the learning rate in the end. Training the 5B top-level requires GPipe which is not supported in this release.

Citation

Please cite using the following bibtex entry:

```
1 @article{dhariwal2020jukebox,  
2   title={Jukebox: A Generative Model for Music},  
3   author={Dhariwal, Prafulla and Jun, Heewoo and Payne, Christine and  
4     Kim, Jong Wook and Radford, Alec and Sutskever, Ilya},  
5   journal={arXiv preprint arXiv:2005.00341},  
6   year={2020}
```

License

Noncommercial Use License

It covers both released code and weights.