Deep Learning Workshop - Project Report
Music Generation via S4

Gal Bezalel[†]

[†]Tel Aviv University, galbezalel {at} mail.tau.ac.il

Abstract

Generating music is an exciting application of sequence modeling. We are utilizing an S4-based architecture called SaShiMi [2] for generation of jazz music. The promise of S4 based architecture to handle long-range dependencies in sequences while capturing it with, potentially, order-of-magnitude less model parameters introduced a straight-forward project idea to work on with limited resources. While the results are not enjoyable per se, we show that the model can, unconditionally, generate at least medium fidelity music. Our findings show that the SaShiMi architecture may not be a good choice for generalized music generation (i.e. a marketable product level), and affirm the notion that obtaining reasonable results with deep learning models in the audio domain is still challenging.

1 Overview

This project is aimed at leveraging the recent advances in sequence modeling made by structured state-space models, and explore how well those architectures can generate music, *preferably using a small number of parameters* ($\ll 10^9$). The base model to be investigated and trained is SaShiMi [2].

2 End product (system from user's perspective)

The ideal end product is straightforward: a user will input a music waveform as a prompt. The system will generate a continuation based on this prompt, akin to a companion musician or a band mate. Alternatively, the user could also enjoy unconditional generation ("improvisation").
Naturally - considerations concerning usability, functionality and performance will come into play:

- From a product perspective, user's prompt need not a be strictly pre-known `.wav` file. Rather, the user should be able to dynamically choose the music source (e.g. from YouTube) and the system will take care of required conversions.
- The backend model might be constrained by number of parameters, usable training data, etc. - and therefore may not generalize well for diverse pieces of music (e.g. different instruments, or genres). In order to compensate for such limitations, the frontend will guide the user towards reasonable prompts.

3 Training and inference schemes

Originally, the following training schemas were considered:

- ◇ **Naive finetuning** - According to the article, training on the YouTubeMix dataset was done for 600K steps; So simply continuing training from the published checkpoint could be done - first, for 150K steps in the hope of showing significant improvement in loss value, and continuing if so. Mixtures of the original pre-training dataset and new FT datasets should be used.
- ◇ **Complete Pretraining** - Training the model from scratch could be feasible (see Compute and storage). We will test this on current maximal configuration (8 S4 blocks) and depending on available resources, consider adding expressive power, e.g. more blocks.

In practice, the **complete training** scheme, effectively replicating the original experiment detailed in the article, but on our own dataset - is the one that fits our end-goal. Moreover, the article demonstrates that *ablated* versions of the model (specifically, a pruned version of only 2 S4 blocks, where expressive power is removed) have shown comparable results with the larger base model. As inference times were significant factor in guaranteeing satisfying user experience, we replicated this experiment as well with our dataset.

Both versions of the model - the full 8 blocks and the ablated 2 blocks - were trained for 1000 epochs, with the default configuration of the original experiment (learning rate $\eta = 0.001$, no regularization). We note that we also trained an additional, regularized version of the ablated model (with a dropout rate of 0.2), but it has shown worse performance and thus we have not included it in our discussion to follow.

Inference can be done conditionally, with a prefix from the validation set, or unconditionally (generate from scratch from the distribution captured in the parameters). The fact that conditional generation was constrained to the validation split of the original training dataset presented a challenge for our ideal end-product: even up to additional data acquisition and preprocessing, our trained model **is not able to generalize to other audio tracks** (even within the genre). Thus, in our end-product demo - we will rely on unconditional generation, and supply conditional generation samples for reference. The upshot being that all generation is done solely with model parameters and no dependency on external data. For generation, we have used the default hyperparameters of temperature at 1.0, and top-p of 1. We found out that lower temperatures, such as 0.2 which are typical when using large language models, generate long, silent samples. Therefore, we did not experiment any further with inference hyperparameters.

4 Datasets

As mentioned in the original article [2], long audio tracks from YouTube can be downloaded and used. While the original YouTubeMix dataset introduced classical piano tracks, we were interested in more complex musical patterns - those that combine multiple instruments (including percussion), and some inherent un-

predictability. This, naturally, gave rise to the idea of using jazz music, played by a big band.

The YouTubeBigBand dataset [1] originates in a 2 hour long Youtube video of smooth jazz music played by a big band. In the spirit of preserving the original experiment's setting, this track was converted from stereo to mono, resampled at 16khz and chunked into 1 minute `.wav` tracks. However, we did not quantized the data into 8 bits, and kept it's original 32 bits quantization. This might have had an effect on our model's performance.

5 Compute and storage

As noted in the original article [2], the autoregressive versions of SaShiMi were trained on a single V100 GPU machine*. This made training on Google Colab feasible (within reasonable costs of the Pro+ subscription). However, with time passing by, in order to accelerate training times, avoid unexpected interrupts and training multiple models in parallel, a high-compute VM instance with A100 GPU was utilized on GCP. Indeed, this resulted in a faster single epoch time for the 8 layers base model (10 minutes on A100 compared to 17 minutes on V100, when it is the only model being trained).

6 Third party tools and models used

The base model to be trained is SaShiMi [2], which is an adaptation of the original S4 model [4] into the audio domain. All S4 descendants are open source. As mentioned in the original repository, the training and generation can be done by invoking the `train.py` and `generate.py` scripts, with a Hydra configuration (either stored in a `.yaml` file under `./configs/` directory, or passed as arguments to the command). The training process and checkpoints are managed using PyTorch-Lightning.

The S4 architecture employs a compute-intensive convolutional Cauchy kernel, for which the repository supplied 2 alternative implementations: a custom CUDA kernel (requires installation), or using PyKeOps. We found that PyKeOps performs significantly faster (up to 10x) on GCP/Colab envs.

The above-mentioned dependencies and additional ones are detailed in the standard `requirements.txt` file. We note that in the original repository, this file was outdated and incorrectly formatted. We provide an updated version of this file that can be installed via `pip` and other Python package managers.

7 Results

Table 1 shows the evaluation metrics output by the `checkpoints/evaluate.py` for our models, as well as for the original model trained on YouTubeMix. Clearly, our experiments have not matched any of the original experiment metrics. This could be explained by the fact that our dataset consists of less samples (2 hours

*It was mentioned that in such setting, training took up to a week.

Table 1: Comparison of evaluation metrics between the original experiment and our experiment.

Test metric	YouTubeMix - 8 layers	YouTubeBigBand - 8 layers	YouTubeBigBand - 2 layers
final/test/accuracy	0.4203284681	0.2712537646	0.2763933837
final/test/accuracy@10	0.9719890952	0.8419079781	0.8469367623
final/test/accuracy@3	0.8351296782	0.5753707886	0.5834547877
final/test/accuracy@5	0.9241486192	0.7081467509	0.7151495218
final/test/bpb	2.063964605	3.186672688	3.148929119
final/test/loss	1.430631161	2.208833218	2.18267107
final/val/accuracy	0.4274106026	0.1958580017	0.2006134242
final/val/accuracy@10	0.9736651778	0.7966675162	0.8028350472
final/val/accuracy@3	0.8423588276	0.4739839137	0.4833323658
final/val/accuracy@5	0.9283464551	0.6271582842	0.6360203624
final/val/bpb	2.029698849	3.658079863	3.611066103
final/val/loss	1.40688026	2.535587311	2.503000021

compared to 4 hours), and, as mentioned earlier, the samples themselves introduce a more complex waveform.

Another interesting finding it that, indeed, the ablated 2 layers version has comparable performance with the full 8 layers version, and in some metrics even beats it.

In the context of our application, supervised learning metrics are not necessarily sufficient in determining whether generated output of the model will sound good (the original article have used Mechanical Turk crowd sourcing for opinionated scoring). However, it indicated that generated samples will not follow the original dataset faithfully.

Indeed, when listening to the generated samples, the musical structure is incoherent: Samples begin with about 2-3 seconds of the big band playing (what could be described as a) song, and then the sample degenerates into some kind of *instruments tuning session*: a high pitch note from the brass section is played constantly, with small changes, and hints of percussion and bass can be heard throughout. The authors mention that:

> Samples generated by autoregressive models can often have "runaway noise", where a sample suddenly degenerates into pure noise. Intuitively, this happens when the model finds itself in an unseen state that it struggles to generalize to, which is often the case for extremely long sequence generation. We found that SaShiMi also suffers from this problem when generating long sequences, and fixing this issue for autoregressive generation is an interesting research direction.

In our generation samples, we haven't encountered pure noise - that is, some resemblance to the original dataset is always maintained. However, indeed, as the generation length grew, the sound became more ambient and incoherent, and the above-mentioned hints became more sparse.

8 Lessons learned

Above all, this project served as a learning practice. We can point out the following lessons:

- ◇ Audio generation is costly, and *good* audio generation is difficult. This is also noted by the original authors. A lot of compromises were made in order to provide results within time and resource constraints; ideally, were this going to be productized - we would have wanted to train on a larger dataset, with higher quality (keeping original sampling rate and stereo channels), and adjust the model accordingly.
- ◇ Despite these challenges, working with out-of-the-box repository has made experimentation with different configurations really easy, and probing interim checkpoints was shown to be a good strategy for deciding whether we should continue with an experiment or not (e.g. when we tried dropout).
- ◇ As previously mentioned, the ablated version is *indeed* on-par with the deeper model. This is an ongoing and exciting research direction [3] that could reduce future costs, as well as generation time while providing high-quality results.
- ◇ Experience gained with infrastructure tools (GCP) is invaluable.

9 Potential extensions (future work)

Naturally, many future action items and ideas come to mind:

- ◇ Naively, working with a larger dataset and seeing whether this suffices to improve metrics and generation results.
- ◇ Hyperparameters grid search:
  - Aside from regularization, trying a larger learning rate is a classic experiment.
  - Since the ablated version has shown promise, an immediate question would be about the "embedding space": what would have happened if we enlarge the S4 block state and/or FF layers width? The default configuration sets `model.layer.d_state=64` as well as `model.d_model=64`. Due to time limitations we weren't able to experiment with doubling those dimensions.
- ◇ "Productize" the model: some careful engineering of the configurations and the `dataloader` module might have resulted in the ability to load any dataset as prompt for conditional generation. However, the results might have been more disappointing due to reliance on our specific training set waveform.

10 Try our model

Our fork is available at https://github.com/galbezalel/s4-dl-workshop. The code will only run in CUDA-enabled environments, with minimum 2GB of GPU memory. We supply both the 8 layers and 2 layers weights under the `checkpoints` directory. Checkpoints are also available via HuggingFace.

10.1 Installation

1. Clone our repository and navigate into it:

```
cd <YOUR_DIR>/s4-dl-workshop;
```

2. Add permissions to the installation script and run it:

```
cd ./tau_workshop;
chmod +x .install_deps.sh;
./install_deps.sh;
```

3. *Optional, for conditional generation*: Set a HuggingFace Access Token and download the YouTubeBigBand dataset:

```
export HF_TOKEN=<YOUR_HF_TOKEN>
cd ./tau_workshop;
python download_dataset.py youtubebigband
```

10.2 Demo - Generation

10.2.1 Via UI

We provide a small application that can generate samples of length between 5 and 60 seconds. The app will employ the 2 layers model for faster generation, however, even the generation of 1 minute of audio on a A100 GPU could take up to an hour. While you wait for your generation to be completed, you can listen to our band's greatest hits. To run the app, simply:

```
 cd ./tau_workshop/app;
python app.py
```

10.2.2 Via CLI

To generate samples via CLI, we refer you to the generation documentation within the SaShiMi README.md file within the repository. The following will use the 8-layers model to generate 30 tracks, each of 10 seconds (a waveform of total 160K steps, at 16Khz) of audio, in the specified `save_dir`:

```
 python -m generate \
experiment=audio/sashimi-youtubebigband \
checkpoint_path=<YOUR_DIR>/s4-dl-workshop/checkpoints/sashimi_bigband_8l.pt
\
n_samples=30 \
l_sample=160000 \
load_data=false \
save_dir=<YOUR_DIR>/s4-dl-workshop/sashimi/uncond-8-layers \
temp=1.0 \
```

10.3 Listen to generated samples

Our app allows you to listen to some of the pieces by our band. All of the samples we generated are available at our dataset on HuggingFace as well.

References

[1] G. Bezalel, "Youtubebigband," https://huggingface.co/datasets/galbezalel/youtube_bigband, 2024.

[2] K. Goel, A. Gu, C. Donahue, and C. Ré, "It's raw! audio generation with state-space models," 2022. [Online]. Available: https://arxiv.org/abs/2202.09729

[3] A. Gromov, K. Tirumala, H. Shapourian, P. Glorioso, and D. A. Roberts, "The unreasonable ineffectiveness of the deeper layers," 2024. [Online]. Available: https://arxiv.org/abs/2403.17887

[4] A. Gu, K. Goel, and C. Ré, "Efficiently modeling long sequences with structured state spaces," 2022. [Online]. Available: https://arxiv.org/abs/2111.00396