January 22, 2022

DEEP NOTE

Karim Kohel, Mohamed Osama

Abstract

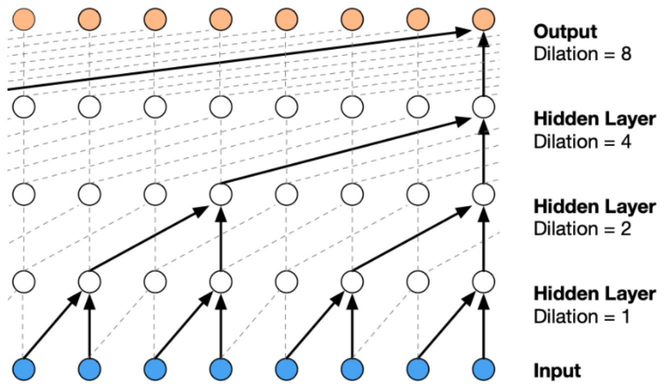
Allowing machines to be indulged in creative fields shows the pinnacle of human-computer interaction. For machines to enter the field and be recognized as a valid competition is another thing. Comes WaveNet, "a deep generative model of raw audio waveforms" (van den Oord and Dieleman, 2016). Letting in the possibility for us to train machines to produce music just as musicians practice their music, but as much as real musicians, the process required time and effort to produce a tolerable output. The model is more than capable of producing real world usable musical scores if trained properly and made easier to try, and this is exactly what we did while testing the model on multiple levels of training with a range of techno music datasets using the python implementation of WaveNet.

WaveNet

arXiv:1609.03499v2 [cs.SD] 19 Sep 2016

WaveNet is the leading model in audio generation when it comes to raw audio samples. Modeling raw audio has been avoided by researchers as it can scale exponentially, and what better way to tackle that problem than to make an exponentially growing receptive field for the convolution layer using various dilation factors as The model is fully probabilistic and autoregressive (van den Oord and Dieleman, 2016).

Using real world samples for training, the model generates and validates the generated output as it goes. For sample generation, each sample is drawn from the probability distribution, then the generated sample is then fed back as input and a new prediction for the next step is made based on the input, thus making output audio sequential in nature as each sample is loosely based on the last one(van den Oord and Dieleman, 2016).



Implementation

A year later (2017) after WaveNet research paper was published Igor Babuschkin started the open source implementation of the network in tensorflow 1 as tensorflow-wavenet on github under the MIT License.

With the help of 31 other open source contributors, the WaveNet tensorflow implementation was packaged in a neat bundle of tools to allow for easy training and audio generation, or so it seemed as it hasn't been maintained for the last 5 years.

How we used the implementation

After recognizing how outdated the Igor implementation was, we were faced with 2 options, either we use the most famous implementation (which was Igor's) and maintain it ourselves by updating code, tools and packages needed to run the model, Or to go fork hunting for the best up-to-date fork of Igor's implementation. We did both, after searching for the best fork to use; we recognized that most people working in the field of audio generation are laser-focused on text-to-speech applications for the model, which made our job much harder and much more interesting. After failing in finding the best implementation suited for music generation we had to use the original one and modify it as much as we need by removing old deprecated tensorflow 1 library calls within the code, matching the drivers and their compatibility with each other and hunting down the old tools used in the implementation for music handling in python to just let it work as intended.

Python Environment

For python virtual environment management we used conda. While most of the dependencies were from official conda repositories, others required the use of pip to find the exact version needed to work.
The environment requires the following modules/frameworks in that specific order to be downloaded:

1. tensorflow-gpu==1.15.0
2. cudatoolkit==10.0.130
3. cudnn==7.6.5
4. librosa==0.5

The last official Tensorflow 1 version had to be 1.15 because unmaintained code in other versions broke the environment.
cudatoolkit 10 had to be exactly 10.0.130 as other versions were not found in conda repositories.
This version of cudnn was the only 7+ version to work with this cudatookit version for some reason.
Librosa 0.5 is required and not version 1 as it contains code-breaking changes.

Sanity check

To test the model after custom modifications to multiple library calls all around the files, we decided to generate a 2 second track. After dealing with dependency errors that came up, the test was complete.

The follwing command was used for training:

```
python train.py --data_dir=techno --logdir=logging
```

where **data_dir** is the directory with all music files in wav format, the **logdir** is the directory for storing log files including the generated model.

while the follwing command was used for generating:

```
python generate.py --wav_out_path=generated.wav --logdir=logging --samples
32000 logging/model.ckpt-15
```

where predictably the **wav_out_path** is the path and name for the generated wav file, **logdir** is still for logging, **samples** flag is the numbers of samples to be generated as the generated file will have an sample rate of 16k which equates the 32k used in the command to the 2 seconds we wished to generate, and finally the model directory and name requires no flag but must be at the end of the command to specify the model used in generating.

Data

The model is designed to train on raw audio samples at any sample rate under the wav file format for audio files. The entire project thesis is based on the uniqueness of our choice of training data, which is Techno-music, as all past experiments focused on natural language processing (Rethage, Pons and Serra, 2018), or ambient music production (Chen and Bordovsky, 2017).

Our collection of 30 to 120 second segments of audio data from techno music files, we trimmed the audio files to only contain the dribbling beats, drums and bass that symbolize the techno genre without the vocals and melodies that might interfere with the signature of the genre.

Training data was collected manually and sequentially through out the project duration, thus letting us train the model several times on several increments of data volums.

To trim unwanted audio durations from training data we used Audacity®(https://www.audacityteam.org) as it was easy to use and can convert .mp3 files to .wav files after editing.

Results

To put the coming magic numbers into context, the model uses steps to show progress in training, could be like epochs, but most probably isn't. On our machines the best we could get was a 1.2 sec average time per step while using a GTX 1660 super with 6GB of VRAM, and an average of 35K steps per 12 Hours of training, which proved to be a very useful metric for us later.

These following results describe a 5-second test audio generated after each training session at a specific step count.

1 Hour of Training Data

10K steps -> 90K steps	Random static noise with a muffled clap or two

3 Hours of Training Data

10K steps -> 30K steps	Random static noise with a muffled clap or two
40K steps	Started producing an accurate sounding drum set accompanied by some incoherent bass sounds with a noticeable level of noise all over the palce
50K -> 90K	produced a dominant hissing sound with continous snares and noise all over the place with underling bass that got worse the more it got to train

6 Hours of Training Data

70K steps -> 80K steps	Random static noise with a muffled clap or two
90K steps	Low volum static noise with heavy bass notes
100K steps	Dominant claps and kicks flowing with a sound of an electric piano playing in the background of some low volume noise
110K steps	A continous heavy bass note with a random clap or two and noise
120K steps	A continous heavy bass note
130K steps	Queit less noisy bass note with some kicks

Bibliography

1. van den Oord, A. and Dieleman, S., 2016. WaveNet: A Generative Model for Raw Audio. [online] Deepmind. Available at: <https://deepmind.com/blog/article/wavenet-generative-model-raw-audio> [Accessed 4 January 2022].
2. Rethage, D., Pons, J. and Serra, X., 2018. A Wavenet for Speech Denoising. 3(1), pp.3,4,5,6,7.
3. Chen, R. and Bordovsky, S., 2017. *Generating Ambient Noise from WaveNet*. [online] Medium. Available at: <https://medium.com/@rachelchen_49210/generating-ambient-noise-from-wavenet-95aa7f0a8f77> [Accessed 15 January 2022].