Music Generation Using WaveNet

Yen-Sung Chen / (Yi-Ting Cheng)

Introduction

The applications of artificial neural networks to music modelling, composition and sound synthesis are not new, but the unprecedented accessibility to resources including data, computational power and superior training method are. We are curious about how we can play with the audio data, and how it is different from the image applications that we have been working on throughout this semester. The model we use in this project is based on WaveNet, a generative model for raw audio, which is proposed by Google DeepMind. Music pieces of different genres and lengths are processed and studied in this work.

Related Works

Recently, there are plenty of works using Recurrent Neural Networks (RNNs) to model music as a sequence of symbols, since they are dynamical systems that incorporate an internal memory, represented by a self-connected layer of neurons. Some of the systems model and generate output at the note level rather than at uniform time steps; e.g., each unit of the input and output layer can represent different pitch. In particular, Long Short-term memory (LSTM) and are extensively employed to learn the structure and characteristics of music since it tackles the error-flow problem better.

Eck and Schmidhuber (2002) are the first to apply LSTM networks to music modelling and generation. The hidden layer of this system consists of two blocks of 8 LSTM cells each, with one block devoted to melody and the other to harmony. The melody block makes recurrent connections to the harmony block. Since then, numerous LSTM-based algorithms have been devised for music-related tasks.

Google DeepMind published WaveNet, which explores raw audio generation techniques. This architecture is able to model distribution over thousands of random variables. The joint probability of a waveform $x = \{x_1, \dots, x_T\}$ is factorized as a product of conditional probabilities as: $p(x) = \prod_{t=1}^{T} p\,(x_t | x_1, \dots, x_{t-1})$. Each audio sample is therefore conditioned on the samples at all previous time steps. The conditional probability distribution is modelled by a stack of convolutional layers.

The convolutional layers are causal; the prediction by the model at time step t cannot depend on any of the future time steps. This could be done easily by shifting the output of a simple convolution by a few time steps. Because models with causal convolutions do not have recurrent connections, they are typically faster to train than RNNs.

One major problem of causal convolutions is that they require many layers to increase the receptive field. We can use a dilated convolution where the filter is applied over an area larger than its length, by skipping input values with a certain step. Stacked dilated convolutions enable networks to have very large receptive fields with just a few layers while preserving the input resolution throughout the network as well as computational efficiency.

In WaveNet, the dilation is doubled for every layer up to a limit and then repeated: e.g., 1, 2, 4, …, 512, 1, 2, 4, …, 512, 1, 2, 4, …, 512.

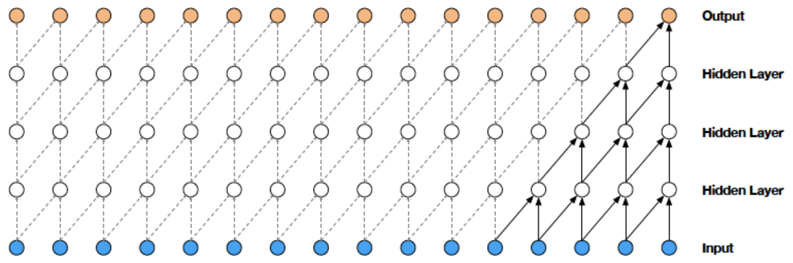

Fig. 1 Visualization of a stack of causal convolutional layers

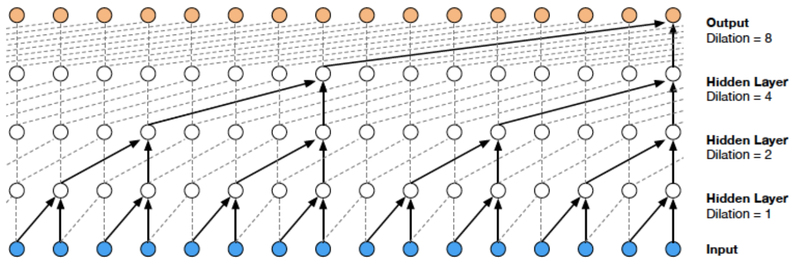

Fig. 2 Visualization of a stack of dilated causal convolutional layers

A softmax distribution is used in WaveNet to model the conditional distributions $p\ (x_t|x_1, \dots, x_{t-1})$ over the audio samples. Since raw audio is often stored as a sequence of 16-bit integer values (one per time step), a softmax layer would need to output 2^{16} = 65,536 probabilities per time step to model all possible values. To relieve the workload but still maintain a certain level of precision, for instance, we can apply a μ-law transformation to the data, and then quantize it to 256 possible values: $f(x_t) = sign(x_t)\frac{\ln\ (1+\mu|x_t|)}{\ln\ (1+\mu)}$, where $-1 < x_t < 1$ and $\mu = 255$. This non-linear quantization produces a significantly better reconstruction than a simple linear quantization where the reconstructed signal sounded very similar to the original one.

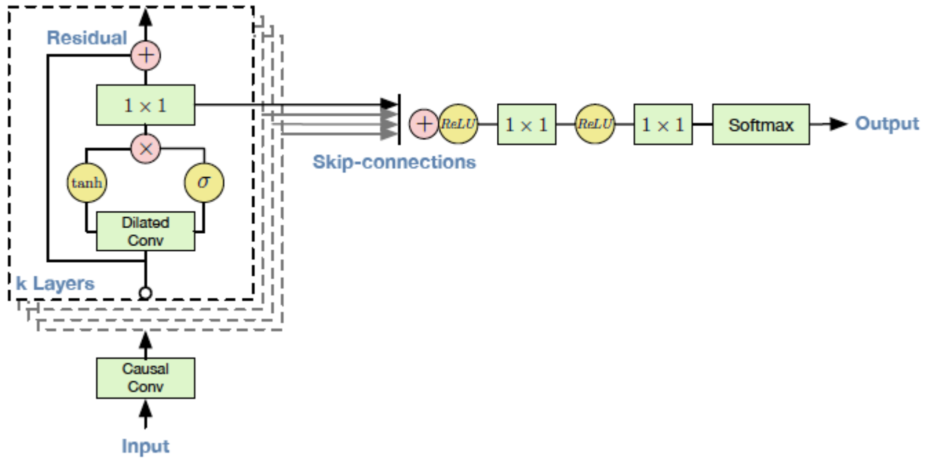

Fig. 3 Overview of the residual block and the entire architecture

Gated activation unit $z = \tanh\left(W_{f,k} * x\right) \odot \sigma(W_{g,k} * x)$, where $*$ denotes a convolution operator, $\odot$ denotes an element-wise multiplication operator, $\sigma(\cdot)$ is a sigmoid function, k is the layer index, f and g denote filter and gate. W is a learnable convolution filter. As can be seen in the overall architecture in Fig. 3, residual and skip connections are used throughout the network to speed up convergence.

Our Results/Discussion

Since the WaveNet aims to deal with raw audio waveform directly, it is natural to input music samples with Waveform Audio File Format (WAV). However, most of the music pieces I obtained are mostly downloaded from Youtube as MPEG Audio Layer III Format (MP3) and subsequently convert to WAV format. Owing to the fact that MP3 is a lossy and compressed format, even if I convert it to WAV, a lossless and uncompressed format, the quality would at most stay the same. This could be one reason for our undesirable results.

At first, we tried to use songs with vocals as inputs, but the results were worse than those with non-vocal music. That is, we can at least generate audio samples that bear some similarities (tempo, tone) to the input using instrumental pieces, but for music with vocals, the resemblance is little. One probable reason is that human voice adds more complexity to the music, and our network is unable to learn the characteristics, and when it tries to generate audio samples, the system instead blurs the non-vocal part with unsuccessful vocal generation.

To improve the quality of generation, we have tried several approaches, from preprocessing the input music to modifying the parameters of the network structure.

Originally, the only preprocessing step in the model that we use is silence trimming, which removes the silence parts from the start and the end of the music. But we have to set different “silence threshold” for each individual music we downloaded since they were all played at different volumes. To solve this problem, we normalized the amplitude of the song while converting it to WAV format so that we can use a single threshold for all the inputs.

We also tried to change the sample rate and the audio channel in the format conversion stage. For example, I reduced the sample rate from 44.1 kHz to 32 kHz, expecting to see the downsampling would help the network to learn better. With the similar idea, I modified the audio channel from stereo to mono, hoping to produce more coherent pieces by relieving the workload for the network. Unfortunately, these attempts did not make noticeable improvement in our generation, so these parameters might not be the deciding factors in training the network for music. Apart from training a full song, I also experimented using multiple single-note file from piano to train the network, but there was no notable result.

There are some parameters of the model that we have played with, like the sample size of each training step. Since we do not have access to powerful computing resources, rather small sizes are chosen to limit the runtime to about 1 sec/step. The number of dilated convolution blocks and the level of quantization for softmax distribution were also modified for optimization purpose. The biggest number of steps for training that we used is 10^5, and most of our experiments were trained more than 10^4 steps. But due to time limitation, we did not further increase the step number.

Not surprisingly, the best results we get comes from the kind of music that possesses obvious tempo and faster rhythm. On the other hand, smooth and soft music tend to be covered by the noise and therefore indistinguishable in the generation audio.

Reference

[1] A. van den Oord et al., *WaveNet: A Generative Model for Raw Audio*, arXiv: 1609.03499, 2016
[2] T. Le Paine et al., *Fast WaveNet Generation Algorithm*, arXiv: 1611.09482, 2016
[3] B. Sturm et al., *Music transcription modelling and composition using deep learning*, arXiv: 1604.08723. 2016
[4] A. Huang et al., *Deep Learning for Music*, arXiv: 1606.04930
[5] I. Babuschkin, github.com/ibab/tensorflow-wavenet

Link to Our Video: https://drive.google.com/file/d/1EyihRRdD0q4P3--WMjbIEBVSSbjvfIOH/view