Midi2Wave Project Report

Gary Plunkett

March 2019

1 Introduction

This project is an implementation of the midi2wave component used in ”Factorized Music Modeling with the Maestro Dataset” [6]. The midi2wave component in [6] is a WaveNet, a convolutional autoregressive neural network capable of outputting raw audio with coherence across tens of thousands of output samples. This WaveNet is conditioned on piano midi data and trained to reproduce corresponding piano audio. In order to realize this WaveNet, [6] also introduced the Maestro dataset, which consists of 172 hours of finely-aligned professional-quality piano audio and corresponding midi data. The main contributions of my project are a set of open-source tools for preprocessing Maestro data for training, and a WaveNet implementation which meets the specifications given in the paper.

As of writing this report, there is still work to be done training the WaveNet to operate successfully in inference mode. With teacher-forcing the output behaves as expected, reproducing audio with quality proportional to model size and time trained. During inference, the WaveNet's output quickly diverges from the manifold of teacher-forced output, degenerating into noise. Additionally, inference-time output is remarkably unresponsive to midi conditioning input, suggesting posterior collapse [4].

2 Literature Review

WaveNets were introduced in (Van den Oord, 2016)[14], building on work using dilated convolutions to generate images on a pixel-by-pixel basis [12]. Stacked, dilated causal convolutions are at the core of WaveNets: By exponentially increasing the dilation size at each convolution layer, the receptive field size of the model grows exponentially with network depth. This allows WaveNets to model sequence dependencies on the order of tens of thousands of samples, an extremely difficult task for RNNs. [14] reported state-of-the-art naturalness in human speech generation, and promising results in music generation. Another important contribution by [14] was the use of a conditioning signal to direct WaveNet output. By providing a speaker identification vector, a single WaveNet was able to reproduce the speech characteristics of multiple speakers.

The Fast WaveNet algorithm for quick inference was introduced soon after the first WaveNet paper. [15] It is a simple memoization algorithm which adds a memory queue to each dilated convolution, so convolution inputs for each layer can be looked-up instead of recomputed.

WaveNets were originally trained on mu-quantized audio, and output categorical distributions corresponding to the 256 potential values of mu-quantization. The paper Parallel WaveNet [13], though focused on the problem of fast WaveNet inference with inverse-autoregressive flows, also introduced using a discretized mix of logistic probability distributions as WaveNet output. The logistic mix distributions can model continuous output, allowing them to model 16-bit audio directly. This technique was again adapted from pixel-by-pixel image-generation networks. [17] Both logistic mixture output and categorical mu-law output can be used without significant difference in audio quality, though the Parallel WaveNet paper mentions that logistic mixture output speeds up early training.

The DeepVoice series of papers detailed end-to-end text-to-speech pipelines, utilizing a WaveNet as the final audio synthesis layer. The original DeepVoice [2] used as conditioning input phoneme type, phoneme duration, phoneme stress, and fundamental frequency. Compared to traditional text-to-speech systems, this was a dramatic reduction in the amount of prepared features required for synthesis. To generate phonemes from text, a multi-component text-to-phoneme model was also introduced. The phoneme conditioning input wasn't fed directly to the WaveNet, but instead preprocessed by a stack of quasi-RNNs. DeepVoice2 [1] followed on the work of DeepVoice by training speaker embeddings to enable state-of-the-art multi-speaker TTS synthesis. DeepVoice3 [16] followed DeepVoice2 by training a network capable of reproducing over 2000 individual speakers (as opposed to DeepVoice2's 108 speakers). Both DeepVoice 2 and 3 were primarily concerned with their text-to-phoneme networks and training speaker embeddings, leaving the WaveNet largely as-is from its incarnation in the original DeepVoice.

Tacotron2 is another text-to-speech system which uses a WaveNet as a vocoder module [18]. The main contribution of this paper is a text-to-speech system trainable on only text and corresponding audio data - no phoneme features required for training. Instead of conditioning the WaveNet on phoneme data, Tacotron2 conditions it on mel-cepstrum frequency coefficients, which are easily calculated from the audio itself. The text preprocessing network learns to to map text to mel-cepstrum data directly.

WaveNets for music generation are notably harder train than text-to-speech networks, due to the need to capture long-term signal information [4]. In music generation networks, WaveNets are often linked together as an autoencoder, where an encoder WaveNet learns to generate conditioning features for a decoder WaveNet. The encoder WaveNet is typically used to capture long-term signal information, such as instrument type, fundamental frequency, and pitch [5] [6] [9]. To capture long-term signal information, the encoder typically operates at a lower sampling frequency, or has a larger, sparser receptive field than the decoder. This is similar to the technique used by audio generation network SampleRNN, which uses multiple tiers of RNNs operating at different frequencies to capture information at different timescales [10]. WaveNet autoencoders have also been used for style transfer: The universal music translator [11] is trained to transform audio between genres, and [7] uses a variational WaveNet autoencoder to transform one speaking voice to another.

3 WaveNet Implementation

My WaveNet implementation began as a fork of the version used in DeepVoice. The only change I made to the residual block structure was multiplying the residual out connection by $\sqrt{0.5}$, a trick suggested by DeepVoice3 to reduce variance early in training. A diagram of the WaveNet is shown in figure 1.

WaveNet output is intentionally left unactivated in order to support outputting both categorical logits, and logistic mixture components. When outputting categorical logits, there will always be 256 output channels for the 256 μ-law audio values. When outputting logistic mixture components, there will be $3 * Nmixtures$ output, as each mixture is specified by the 3-tuple (logit, mean, log variance).

Categorical logits are transformed into a probability distribution via softmax, and the model loss is the negative log-likelihood of observing the true μ-law value given the probability distribution. When sampling from categorical output I use the gumbel-softmax trick, due to its speed and the ability to backpropagate through samples. [8]

3.1 Discretized Logistic Mix Output

The loss for logistic mix components is the negative log-likelihood of observing the true μ-law value given those components. The form of the logistic mixture components are detailed in the PixelCNN++ paper [17]. The different mixture components - logit, mean, and log variance - are activated separately. Logits are softmax'd to create a probability distribution, and means are tanh'd to move the range of output values from [0, 255] to [-1, 1], since using a 0 mean simplifies the calculations required. The log variance is run through a sigmoid and multiplied by -7. Because a mixture component represents a p.d.f., its maximum value must be kept below 1. A mixture component with log variance -7 has a maximum value of 0.98, so -7 is chosen as the minimum log variance. A maximum log variance of 0 is used because at that point the mixture component is already essentially a uniform distribution. Contrary to my implementation, the PixelCNN++ implementation allows the log variance to take any positive value, and is clamped at -7. However, both methods of capping lg variance are fine, as little difference exists between mixture components with positive log variance.

The c.d.f. of the mixture component defined by PixelCNN++ is, conveniently, the sigmoid function. Sampling from a mixture component is as easy as uniformly sampling from the corresponding inverse sigmoid function.

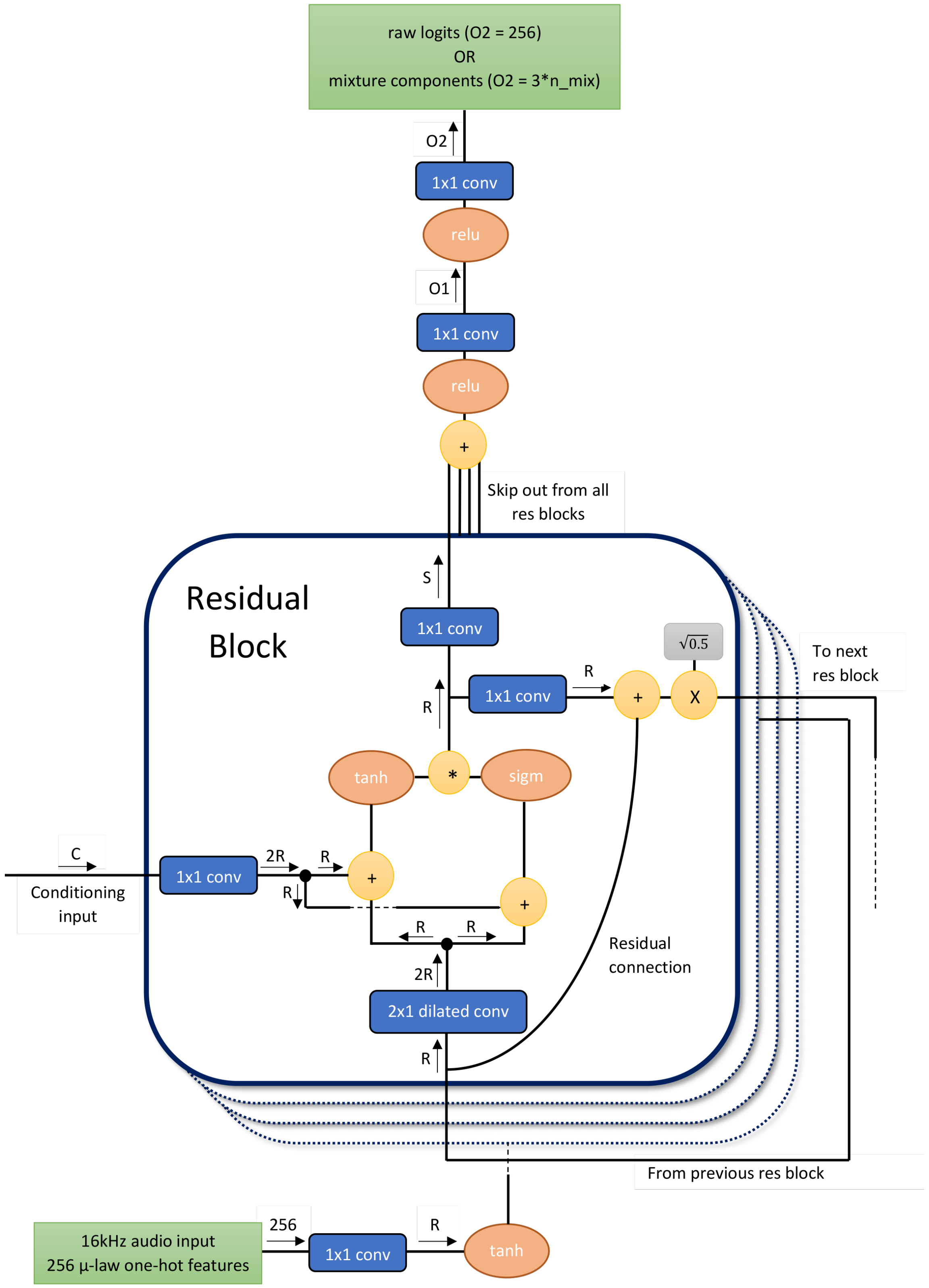

Figure 1: Diagram of the WaveNet used in all experiments.

3.2 WaveNet Autoencoder

The paper's midi2wave module actually uses two WaveNets: The usual synthesizer WaveNet which performs audio autoregression, and a "context WaveNet" which uses midi roll features instead of audio samples as forward input. Midi features are fairly sparse, typically tens of note onsets per 1000 samples, so the context WaveNet functions to spread information from midi features across time. The context WaveNet feeds its output into the conditioning input of the audio WaveNet and is trained by backpropagation through this connection. Because midi features are 250Hz and audio is 16kHz, conditioning input is upsampled by repetition to 16kHz.

The context+audio WaveNet configuration can be viewed as an autoregressive autoencoder, with context WaveNet output as the latent code. In addition to the Maestro paper specifications, I added support for turning the autoencoder into a discretized variational autoencoder, as suggested by [4]. In this configuration, the context WaveNet outputs one-hot vectors, and an L2 regularization term in the loss encourages the distribution of one-hots to be uniform. This makes the context WaveNet output as diverse as possible.

4 Experiments and Results

Each WaveNet configuration I experimented with has been able to reproduce audio when using teacher forcing, but without teacher forcing audio has always devolved into quiet noise or loud piano-like noise. My testing procedure was to generate 4 second audio clips, using teacher forcing for the first 2 seconds, then switching to autoregressive mode for the remaining 2 seconds. This was intended to help the autoregressive module by giving it a history of 'good' samples to draw from, instead of having to start from silence. Whether the the WaveNet generated quiet noise or loud piano-like noise was dependent on the audio preceding it, showing the autoregressive module could predict whether or not it was supposed to be 'on' or 'off' depending on the preceding samples. The piano-like noise shows the audio WaveNet learned to create the formant shape of piano notes, but didn't learn to sustain formant shapes to maintain coherent notes. Midi information from the encoder WaveNet is available to the decoder WaveNet and could provide the information needed for note coherence, but the decoder WaveNet appears to learn to ignore that information and do all autoregressive work itself. This is a known problem for autoencoders with decoders powerful enough to model the underlying data alone, and is called posterior collapse [4]. This section begins with discussing the most recent experiment aimed at preventing posterior collapse, as this is by far the most likely culprit as to why the WaveNet can't operate in inference mode. Following this I discuss results from scheduled sampling experiments, then investigate the the properties of midi and audio signals at the point where they're added together in the WaveNet.

4.1 Training Data Details

Midi data is in a "piano-roll" representation, with 88 note channels and 1 pedal channel for every timestep. Note channel values are nonzero if there was a note onset that timestep, and its value is set to the velocity of the note onset. Pedal values are nonzero if the pedal was depressed at that timestep, with a value equal to to the depression. Velocity and pedal values were normalized to the range [0, 1]. For the training runs below, pedal information was omitted from the midi data since the Maestro paper reports doing so, though in the future I think this would be useful information to incorporate.

Audio data is μ-quantized audio values, one integer per timestep. Additionally, I silence audio before the first midi onset, which should help the WaveNet learn how to "turn on" from silence in the case of a midi onset.

Audio and corresponding midi data is sampled randomly from the entire training set. Because the training set has over 130 hours of audio data, training in terms of epochs doesn't make as much sense as it would for a smaller dataset. Instead, I talk about training in terms of seconds of audio seen. I used a batch size of 1, with an audio length as high as I could fit onto the GPU, typically 5-10 seconds. The Maestro paper reports training with a batch size of 1, with 20 second training samples, so I matched this as best as possible. I used a learning rate of 1e-3, and an Adam Optimizer with default PyTorch β values.

4.2 Discretized-VAE WaveNet

To prevent posterior collapse in WaveNet autoencoders, [4] suggests turning the autoencoder into a discretized variational autoencoder, and encouraging the latent code obey a uniform distribution. They propose discretization because it gives users a way to easily manipulate the audio WaveNet without an encoder. The uniform distribution encourages encoder output to be as diverse as possible, making it harder for the decoder to ignore the encoder signal.

As of writing this, I've had time for one long training run with a discretized variational autoencoder. This WaveNet again suffered from posterior collapse, although the character of its piano-like noise was noticeably altered (not in a positive or negative way, just different). This WaveNet autoencoder was trained on slightly over 1 million seconds of audio data. The midi encoder WaveNet was identical to the one used in the Maestro paper, using two residual blocks with 6 layers each. The receptive field size of the decoder is 254 samples, which is slightly over 1 second long with 250 Hz midi input. As per the Maetsro paper, only note onset velocities were used as forward input. The dimensionality of each residual block was 88, the same size as the midi information. Midi output was transformed into a 512-dimensional one-hot vector before being input to the audio decoder WaveNet.

The audio decoder WaveNet used two residual blocks with 11 layers each, which makes a receptive field size of a little over 0.5 seconds. Its residual block size was 32, and it output 10 logistic mixtures. The loss function used was the same used detailed in section 3.2 of the Posterior Collapse paper: $L = NLL - v * L_{diversity}$. The diversity loss began around 900, decreased to 6 after 10k training seconds, and further decreased to 5e-3 after 1 million training seconds. The negative log-likelihood began at 6.467 and decreased to 2.046, as averaged over the last 2000 training seconds, resulting in fairly high-quality decoder output.

4.3 Encoder Signal Analysis

I ran some basic analysis on the midi conditioning signal and the audio WaveNet residual block signal, at the point where the two are summed in the residual block. this is immediately before the nonlinearity in the residual block, after the conditioning signal is transformed by the conditioning input convolution to the residual block size. With one-hot conditioning input this convolution function as an embedding with backprop capabilities. I looked at the magnitude of the midi signal, the magnitude of the residual block signal, and their cosine similarity. If the cosine similarity between the residual block signal and the conditioning signal is constant, this would be indicative that the midi signal isn't adding any information to the residual block.

The model investigated was the already trained discretized-VAE detailed in the previous section. The average cosine similarity across time and residual blocks for 100 seconds of audio was 0.3644, so on average the two signals were more aligned than not. The midi signal amplitude was suspiciously constant to within a magnitude of 0.002, however. To show that the midi signal was actually constant, and not rotating around residual block space with a constant magnitude, I looked at the average cosine deviation of midi signals from the average midi signal. This value was 0.0025, proving that the conditioning signal seen by the decoder WaveNet was essentially constant no matter the midi signal. The decoder WaveNet's conditioning signal input convolution has to be responsible for this, since the midi module's output was outputting nearly uniformly distributed one-hots. However, I suspect the root cause is the diversity loss meant to guide the midi encoder to outputting a uniformly distributed signal. Because the decoder can model the audio on its own, the encoder isn't very pressured to output a midi distribution, but it is pressured to output a uniform by the diversity loss. I think it very likely the encoder module learned to output a random uniform distribution regardless of the midi input signal. In this case, the conditioning input convolutions would certainly learn to throw away the midi encoder's information!

I also suspect restricting the latent code to 512 possible values is too much of a reduction in information representation. The signal output by the midi encoder WaveNet should carry information about what notes were recently played, the velocity of the note strikes, and how long ago the notes were played. With 88 possible notes, limiting the model to 512 output values compresses the signal information quite a bit. Using continuous instead of discrete decoder output makes more sense for this task in particular since conveying how long ago a note press occurred is important. The mapping of encoder output to a uniform distribution is a good idea for promoting signal diversity, but I need to do more hyperparameter tuning to ensure the audio loss signal is greater than the code diversity loss signal.

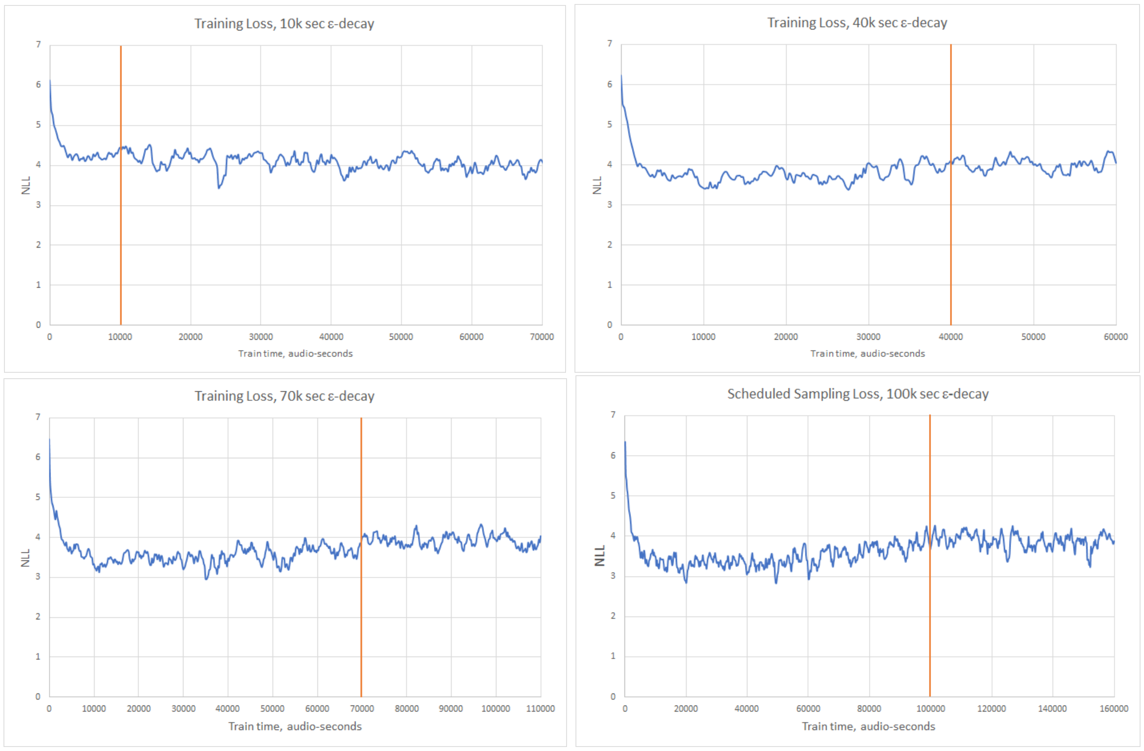

Figure 2: Train loss using scheduled sampling for different ϵ-decay rates.

Even if the encoder module outputs a useful signal, there is still opportunity given to the audio WaveNet to transform/attenuate the decoder signal via the 1x1 convolutions applied to the conditioning input. In future experiments with a variational autoencoder WaveNet I plan to remove the decoder's conditioning input convolution entirely, and use the latent code directly as conditioning input for each layer. Biasing every layer with the same signal has been shown to work in practice by DeepVoice 2 and 3, and should further increase the midi signal strength.

4.4 Scheduled Sampling Experiments

At inference time, an autoregressive models' output may diverge from the manifold of the underlying distribution, putting the model into a feedback loop of outputting nonsense. The audio WaveNet's collapse into noise in the absence of teacher forcing was suspected to be due to this phenomena. Scheduled sampling is a training method for autoregressive models where teacher-forced model predictions on are used as training data. By showing the model its own predictions at train time, the manifold of observed data increases to include model predictions, leading to higher model stability at inference [3]. A number of scheduled sampling experiments were conducted, none of them showing success.

My scheduled sampling procedure was as follows: First, train time predictions $y_{predictions}$ were generated by a forward pass, using midi input x and teacher audio y. Next, for every timestep, a random number from $[0, 1)$ is generated. These random numbers are compared against ϵ, which is some value between 0 and 1 used to indicate the likelihood of using a true y sample or a predicted one. y_{new} is created by combining y and $y_{predictions}$, using the value from y if the random number for that timestep is $< \epsilon$, and using the $y_{predictions}$ value if that timestep's number is $> \epsilon$. The model is trained on the resulting y_{new}.

[3] recommends slowly decreasing ϵ over time, gradually introducing more prediction samples to the data. I trained four identical models with different linear epsilon decay rates. The model trained was a WaveNet autoencoder with the same architecture as the discretized-VAE above (so minus the quantization module and diversity loss). The latent code size was reduced from 512 to 89 since I was using continuous encoder output. Also, unlike the discretized-VAE I included midi pedal information. Figure 2 shows train-time negative

log-likelihood loss for 4 different scheduled sampling schedules. The ϵ decay began as soon as training began. It was decreased from 1 to 0.1 over the course of the decay length. The end of decay is indicated by an orange line on the plots. Also, I introduced a 1-in-10 chance of not using scheduled sampling, and training the model on entirely true audio.

If scheduled sampling was to improve model performance, then after increasing epsilon decay to a certain point, the model should see improvement in its training loss, even if it never returns to the same minimum as before. However in my experiments, each model ended up at the same average loss after ϵ decay, and never improved. Loss improvement never happened for any training run, no matter how long I let it train after *epsilon* was done decaying. This is especially noticeable in the 10k seconds decay plot, where the model is never given time to drive its loss below the scheduled sampling loss.

I initially conducted some experiments with multiple-forward-pass scheduled sampling, to see if the model could learn from the predictions of predictions and thus expand the manifold of observed samples even more. However, this was unsuccessful, and model quality continued to decrease as I increased the number of scheduled sampling prediction loops.

There is a good reason why scheduled sampling was unlikely to fix the WaveNets' bad autoregressive output: WaveNets are able to recover from autoregressive output divergence due to conditioning input. TacoTron2 [18] reports that sometimes their WaveNet audio will diverge from the true audio manifold and devolve into noise, but that audio quickly recovers due to the power of the mel-cepstrum conditioning signal guiding WaveNet audio. Because my audio WaveNet was completely unresponsive to the midi conditioning signal, I think this shows the root problem is with the midi signal, and not with inference-time divergence from true audio. Additionally, scheduled sampling was introduced to increase the performance of an already functional autoregressive model, not to fix its functioning completely.

4.5 Discretized Logistic Mixture vs Categorical Output

Discretized logistic mixture components were used in all experiments shown in this paper. With categorical output, WaveNet audio collapse to complete silence without teacher forcing. With logistic mixture output, the WaveNet would collapse into quiet noise or loud piano-like noise, depending on the amount of activity immediately preceding audio. Also, the Maestro paper uses logistic mixture output, so there was motivation for using it in the interest of copying their implementation.

5 Conclusion

Modeling music with WaveNets has proven difficult due to having to fight posterior collapse. This doesn't seem to be as much of a problem for speech generation networks, where conditioning information is much more localized (what is the phoneme occurring at this instant vs. what notes were played 2 seconds ago). However for music, the audio WaveNet will learn to do all the work itself if care isn't taken to provide the audio WaveNet with a conditioning signal it can't ignore.

References

[1] Sercan Ömer Arik et al. "Deep Voice 2: Multi-Speaker Neural Text-to-Speech". In: *CoRR* abs/1705.08947 (2017). arXiv: `1705.08947`. URL: `http://arxiv.org/abs/1705.08947`.

[2] Sercan Ömer Arik et al. "Deep Voice: Real-time Neural Text-to-Speech". In: *CoRR* abs/1702.07825 (2017). arXiv: `1702.07825`. URL: `http://arxiv.org/abs/1702.07825`.

[3] Samy Bengio et al. "Scheduled Sampling for Sequence Prediction with Recurrent Neural Networks". In: *CoRR* abs/1506.03099 (2015). arXiv: `1506.03099`. URL: `http://arxiv.org/abs/1506.03099`.

[4] Sander Dieleman, Aäron van den Oord, and Karen Simonyan. "The challenge of realistic music generation: modelling raw audio at scale". In: *CoRR* abs/1806.10474 (2018). arXiv: `1806.10474`. URL: `http://arxiv.org/abs/1806.10474`.

[5] Jesse Engel et al. "Neural Audio Synthesis of Musical Notes with WaveNet Autoencoders". In: *CoRR* abs/1704.01279 (2017). arXiv: `1704.01279`. URL: `http://arxiv.org/abs/1704.01279`.

[6] Curtis Hawthorne et al. "Enabling Factorized Piano Music Modeling and Generation with the MAESTRO Dataset". In: *CoRR* abs/1810.12247 (2018). arXiv: 1810.12247. URL: http://arxiv.org/abs/1810.12247.

[7] Wen-Chin Huang et al. "Refined WaveNet Vocoder for Variational Autoencoder Based Voice Conversion". In: *arXiv e-prints*, arXiv:1811.11078 (Nov. 2018), arXiv:1811.11078. arXiv: 1811.11078 [eess.AS].

[8] Eric Jang, Shixiang Gu, and Ben Poole. "Categorical Reparameterization with Gumbel-Softmax". In: 2017. URL: https://arxiv.org/abs/1611.01144.

[9] Jong Wook Kim et al. "Neural Music Synthesis for Flexible Timbre Control". In: *CoRR* abs/1811.00223 (2018). arXiv: 1811.00223. URL: http://arxiv.org/abs/1811.00223.

[10] Soroush Mehri et al. "SampleRNN: An Unconditional End-to-End Neural Audio Generation Model". In: *CoRR* abs/1612.07837 (2016). arXiv: 1612.07837. URL: http://arxiv.org/abs/1612.07837.

[11] Noam Mor et al. "A Universal Music Translation Network". In: *CoRR* abs/1805.07848 (2018). arXiv: 1805.07848. URL: http://arxiv.org/abs/1805.07848.

[12] Aäron van den Oord et al. "Conditional Image Generation with PixelCNN Decoders". In: *CoRR* abs/1606.05328 (2016). arXiv: 1606.05328. URL: http://arxiv.org/abs/1606.05328.

[13] Aäron van den Oord et al. "Parallel WaveNet: Fast High-Fidelity Speech Synthesis". In: *CoRR* abs/1711.10433 (2017). arXiv: 1711.10433. URL: http://arxiv.org/abs/1711.10433.

[14] Aäron van den Oord et al. "WaveNet: A Generative Model for Raw Audio". In: *CoRR* abs/1609.03499 (2016). arXiv: 1609.03499. URL: http://arxiv.org/abs/1609.03499.

[15] Tom Le Paine et al. "Fast Wavenet Generation Algorithm". In: *CoRR* abs/1611.09482 (2016). arXiv: 1611.09482. URL: http://arxiv.org/abs/1611.09482.

[16] Wei Ping et al. "Deep Voice 3: 2000-Speaker Neural Text-to-Speech". In: *CoRR* abs/1710.07654 (2017). arXiv: 1710.07654. URL: http://arxiv.org/abs/1710.07654.

[17] Tim Salimans et al. "PixelCNN++: Improving the PixelCNN with Discretized Logistic Mixture Likelihood and Other Modifications". In: *CoRR* abs/1701.05517 (2017). arXiv: 1701.05517. URL: http://arxiv.org/abs/1701.05517.

[18] Jonathan Shen et al. "Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions". In: *CoRR* abs/1712.05884 (2017). arXiv: 1712.05884. URL: http://arxiv.org/abs/1712.05884.