Speech Compression and Generation with a Variational Autoencoder

Jamie Pond, *MSc Sound And Music Computing*
(ec20768@qmul.ac.uk)

Queen Mary University of London, London, UK

Here we will examine the creation of a variational autoencoder which can be used to both compress and generate audio from the FSDD dataset [1]. This work has been inspired and guided by the work of Dr. Valerio Velardo (University of Huddersfield) [2] [3]. As a result of limited GPU availability as a result of the COVID-19 pandemic, not as many iterations and variations on hyperparameters and structures have been trialed, but in this paper we will examine the success and drawbacks of the given model. The lowest bitrate achieved was 0.91 kb/s.

0 INTRODUCTION

The main focus of this paper is to create a performative Variational Autoencoder (VAE), which is able to compress audio data into a compact latent space, which has the use case of offering low-bandwidth voice communication, which has implications in reducing the data cost in speech communication for countries with low-bandwidth data infrastructure, but also increasing the throughput possible in countries with preexisting good infrastructure.

This echos the recent work of Google AI's Lyra [4], which has achieved bitrates as low as 3kbps with very reasonable intelligibility. The results achieved here do not reflect the same audio quality, but offers an alternative architecture and approach.

To train this model, spoken digits from the FSDD dataset [1] were chosen to train the model as they represent the kind of audio that would be relevant to the use-case that is being investigated. The dataset is like an audio equivalent to the MNIST dataset.

There are two component models which need to be trained here; the encoder and the decoder. The encoder is an important part of the VAE, as this is what is leading up to the latent representation, which is effectively our 'compressed' form of the data. This step is equivalent to 'zipping' on a traditional lossless compression algorithm (this is of course a form a lossy compression like the mp3 codec).

This then presents us the latent representation in the centre of our VAE, which we extract at the 'bottleneck' of the model. This is where we can 'save' the latent representation, and then perhaps transmit it over email or in streaming packets.

This then leads on to the decoder. This takes the latent representation and infers the full spectrogram, which can then be converted back into the raw audio form. The raw audio output of this model is still quite robotic sounding, but this may be a function of the spectrogram to audio conversion and the loss of phase information in the process.

1 IMPLEMENTATION

1.1 Pre-Processing

All of the raw audio data from the FSDD dataset had to be preprocessed so that we could interpret the data as 2D spectrograms (256×64), as that is more computationally practical than an end-to-end approach, though we will discuss the benefits of attempting an end-to-end approach later.

The audio was transformed into log spectrograms, with the use of the Librosa library. In order to assist the audio reconstruction from spectrograms at the end of the minimum and maximum values of the audio were also stored which contributes to an extra two floating point numbers in the 'encoded' representation aside from the pure latent representation.

1.2 Loss Function

At this point it is important to mention the loss of the system. This will come in the form of Root Mean Square Error between the input spectrogram and the reconstructed spectrogram i.e.

$$RMSE = \sqrt{\Sigma_{i=1}^{n} \frac{(\hat{y}_i - y_i)^2}{n}}$$

However, since we are using a variational autoencoder which encodes the latent space as a multivariate normal distribution, it is also wise to implement a (closed form) Kullback-Leibler divergence loss term. This measures the difference between two probability distributions which penalizes observations where the mean and log variance vectors differ significantly from the parameters of a standard normal distribution.

$$D_{KL}(N(\mu,\sigma)||N(0,1)) = \frac{1}{2}\Sigma(1 + log(\sigma^2) - \mu^2 - \sigma^2)$$

Here the distribution to be tested and the standard normal distribution are given as arguments.

This means the overall loss function for this VAE is:

$$loss = RMSE + D_{KL}$$

1.3 Architecture

In this section we will examine the architecture of the system. As mentioned before the input layer to the encoder has dimensionality (256×64), which corresponds to 256 frequency bins and 64 time stamps. This is a result of the radix-2 STFT in the pre-processing stage, so if we wanted to reduce or change this, at least in the frequency bin dimension, we would need to have this in powers of two and adjust the pre-processing accordingly.

This model is built up with a number of convolutional layers. The template for each convolutional block is a convolutional filter, a ReLU activation function and a batch normalization.

In the final iteration of the model, the encoder had a total of 5 of the convolutional blocks described above with various strides and filter sizes. This is then followed by the 'bottleneck' of the VAE, which is where our latent representation will come out. To do this we need to flatten the data, which is then passed to a dense layer. The dimensionality of the dense layer is defined by a predefined hyperparameter. In this model the output latent representation of each clip is a (1×128) array. During the flattening phase, we also need to store the dimensionality of the data before the bottleneck, so this can be undone in the decoder. In this model, a 3 integer array is stored to be able to reconstruct the dimensionality of the data in the decoder. This does not contribute to the compressed representation since this can be set in a way that the encoder and decoder 'agree', and will only have certain dimensionality.

At this point the compressed/latent representation can be stored or transferred over the internet. It will then need to be decoded. The latent representation is given as the input to the decoder, which is a symmetric opposite to the encoder. This means that it will have the same number and types of layers as the encoder but in reverse. It will then output the (256×64) spectrogram reconstruction, which can then be turned back into raw audio using the Librosa library.

2 DISCUSSION

Not all of the encoded and decoded audio is intelligible as what it is supposed to be, however, most of them are, which implies that the encoder has been able to learn the essential qualities of the clip and pack them into the latent space. This means that the encoder has been able to learn the pitch and timbral qualities of the input. This thus implies that the decoder is able to interpret that same latent representation and recreate a suitably interpretable spectrogram.

There are a number of ways in which the model could be improved, and many avenues to explore in attempting to improve it.

For example, as a result of the limited compute capabilities available, the model took 17 hours to train on the entire dataset with a suggested 150 epochs [2]. This limited the scope for being able to experiment much with hyperparameters and architectures, however, the results achieved so far are promising. Following on from this, as a result of the long time to train, this has limited the ability to run trials with a good sized testing set. Though a small testing set was created with good results, a bigger testing set would surely help to improve the reliability of the results interpreted here.

It would be interesting to try and reduce the number of trainable parameters in the model, which is currently 2,371,777. With a smaller number of trainable parameters, this would not only decrease training time, which would help in creating more, and higher quality iterations of the hyperparameters, but also decrease the computational cost for encoding and decoding, which will be very important if this system were to run in real time, especially on limited hardware like a smart phone.

2.1 Phase Reconstruction

What would really help the overall structure to be more convincing is to be able to reconstruct the phase of the model, which will help the voice to sound less robotic. It might be proposed that we could perform two sets of VAEs, one of which does the same as described above and accurately encodes magnitudes, and the other may be used to encode phase. This can then be used in the spectrogram reconstruction. Else, a deep learning model could be implemented or designed that is better able to infer sensible phase information from only the magnitude spectrogram data.

2.2 Comparison to Google Lyra

Google Lyra is an excellent speech codec, that differs to this one in a number of ways, including the fact that it uses the Mel spectrogram. This would make for another good possible improvement on this model, since this examines the frequency data in a way which is more akin to the human hearing system. Lyra also uses WaveNet technology in order to help the reconstruction sound more natural, which may be an area to develop for this assignment.

2.3 End-to-End Processing

It may be useful to train the model on an end-to-end architecture in the future, which would be beneficial, as the model, with a suitable architecture, would be able to learn the features for itself, and pick out the features most salient for the task at hand [5].

However, this would be very computationally expensive, and would require much higher training times, that would be outside the scope of this assignment, but it would be interesting to see if the learned features of an end-to-end approach may be more suitable for speech compression task, even more so than feeding it a Mel spectrogram as features.

3 CONCLUSION

In this assignment we have examined how to create a speech compression algorithm, trained on the FSDD dataset, which results in a 14% reduction in file size from the 8kHz sampling rate audio provided, achieving a bitrate of 0.91 kb/s. This is less than 3$\times$ the bitrate of Google Lyra's cutting edge performance, however, the sound quality is not as good, yet and there are still a number of issues still to be resolved such as perhaps remembering phase, inferring or reconstructing phase information in a more meaningful way. It would also be very good to get a larger validation set in order to show the pure performance of the model, and also to set up the model to accept blocks of input, such that it can start to process audio in real-time, so that it could perhaps be used in a voice-calling setting.

4 ACKNOWLEDGMENT

This assignment was completed as a part of the ECS7013P Deep Learning for Audio and Music module at Queen Mary University of London, taught by Dr. Huy Phan. The author would like to thank Dr. Phan for his outstanding teaching and support throughout the module, and as well to all of the demonstrators during their invaluable and expert support in the lab activities.

Their dedication to, and expertise of the field of Deep Learning has been instrumental in inspiring the author to achieve their best, and to help develop their skills as a mathematician, engineer and programmer.

5 REFERENCES

[1] Z. Jackson, "Jakobovski/free-spoken-digit-dataset," URL `https://github.com/Jakobovski/-spoken-digit-dataset`, original-date: 2016-06-21T09:46:20Z.

[2] "Sound Generation with Neural Networks," URL `https://www.youtube.com/watch?v=Ey8IZQl_lKslist=PL-wATfeyAMNpEyENTc-tVH5tfLGKtSWPpindex=1`.

[3] V. Velardo, *musikalkemist/generating-sound-with-neural-networks* (2021 Apr), URL `https://github.com/musikalkemist/generating-sound-with-neural-networks`.

[4] "Lyra: A New Very Low-Bitrate Codec for Speech Compression," URL `http://ai.googleblog.com/2021/02/lyra-new-very-low-bitrate-codec-for.html`.

[5] S. Dieleman, B. Schrauwen, "End-to-end learning for music audio," presented at the *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 6964–6968, [Online]. Available: 10.1109/ICASSP.2014.6854950, ISSN: 2379-190X.

THE AUTHOR

Jamie Pond

Jamie Pond is a software developer at PresentDayProduction and an MSc Sound and Music Computing at Queen Mary University of London student, where his research interests include audio and music applications of signal and array processing, C++ programming and design and development of augmented instruments. From 2017 to 2020, Pond was an undergraduate student at the London College of Music, where he studied Music Technology with a focus on mixing and mastering. During this time, Pond has spent many hours working in commercial recording studios in and around London, including Cosmic Audio in Epping, and RYP Studios in Harrow. In the last two years, Pond has started developing audio plugins with PresentDayProduction, with an aim to create fun and engaging products that creatives resonate with. Pond was also the Vice-President of the Audio Programming Society at UWL, where he organised evening lectures with guest speakers such as Josh Hodge of The Audio Programmer and Professor Eric Tarr