ML quantisation of Codec 2 Vocoder Features

January 26, 2024

The goal of this study is to explore quantisation of Codec 2 features using ML techniques.

It is useful to frame our expectations. There is significant information in a mel spaced log spectral vector, and it will take a finite amount of bits to quantise it. ML won't change this. Where ML can help is better transforms and time prediction over large windows. For example finding non-linear dependencies and correlations and reducing the dimension of the bottleneck required to be quantised. Better transforms and time series prediction than possible with linear techniques. Combination of spectral information with pitch and voicing features will save some bits if they are correlated with the spectral information. A multistage VQ will work better with a small dimension, as the residual errors tend towards Gaussian.

However this will have it's limits for our use case where the quantisation window must be kept to around 100ms. ML isn't magic - it will take more information to quantise 8 x 10ms vectors than 1 x 10ms (but not 8 times as much).

Is running the VQ training (e.g. VQVAE) inside the ML system worth it? The literature cites several practical problems [2], such as the rate the VQ adapts compared to the rest of the network, managing exponential weighted moving average (EWMA) updates that attempt to zero out codebook entries, and possibly poorer perf than traditional k-means. Traditional VQ works pretty well and can extract some non-linear dependencies. The approach tried here is to simulate quantisation in the training loop using small amounts of noise injection to ensure the latent space is well behaved. Then, the VQ can be designed on the training data in the bottleneck. The principle is to use traditional DSP/quantisation where possible, leaving ML to what it does best.

Planned experiments:

1. Experiment 1: Goal is to demonstrate dimensionality reduction. Build an autoencoder for 1 vector. Try different architectures. Determine bottleneck dimension for $1dB^2$ distortion. Try MSE and weighted linear loss functions. Determine if we can get any dimensionality reduction, as this would make VQ much easier (assuming VQ latent space is well behaved).

2. Experiment 1a: Goal is to demonstrate scalar linear quantisation $< 1dB^2$ at any bit rate, and "shape" the latent space. Add noise to bottleneck to

simulate uniform quantisation.

3. Experiment 1b: Goal is to apply VQ to latent spaces. Visualise latent space somehow. Use a traditional VQ to quantise the bottleneck vectors, compare bit rate to 1a.

1 Experiment 1

A simple autoencoder network was designed consisting of 4 FC layers with a $tanh()$ non-linearity at the bottleneck. The training material was dimension $k = 20$ smoothed, unit energy, mel spaced log vectors $\mathbf{b}$. Only one $\mathbf{b}$ vector was considered at a time (no time based transformations). When trained with a SD loss function 9 a dimension $d = 10$ there was a floor of around $1.3dB^2$ (larger than the target $1dB^2$) distortion. Reasonable results (based on visual inspection) were also obtained using a weighted linear loss function 4.

It was unclear if the latent space was suitable for quantisation. To investigate this, a small amount of noise was added to simulate the effect of uniform quantisation over the $[-1, 1]$ dynamic range of the $tanh$ function placed at the bottleneck. This noise also encourages the network to converge on a latent space that behaves well in the presence of small quantisation errors.

Consider a uniform quantiser with s discrete levels applied to the interval $[a, b]$. The reconstruction levels are given by:

$$r_i = a + \frac{b-a}{s}(i + 0.5), \quad i = 0, 1, ..., s-1 \tag{1}$$

It can be seen that each level is $\frac{b-a}{s}$ from the next. The quantisation error for any input x will be uniformly distributed over the interval $[-\frac{(b-a)}{2s}, +\frac{(b-a)}{2s}]$. For the $tanh()$ function the interval $[a, b] = [-1, +1]$, and the quantisation error will be distributed over $[-\frac{1}{s}, +\frac{1}{s}]$.

The variance of the noise from a uniform distribution over the interval $[c, d]$ is given by:

$$\sigma^2 = \frac{(d-c)^2}{12} \tag{2}$$

The variance of our quantisation noise is therefore:

$$\sigma^2 = \frac{1}{3s^2} \tag{3}$$

Table 1 presents uniform quantisers for a range of step sizes. This gives us a baseline to compare more sophisticated quantisation techniques. For example a 5 bit $s = 32$ step uniform quantiser and $d = 10$ bottleneck would require 50 bits/frame for $\sigma = 0.018$ distortion. We hope to improve on that that with vector quantisation (VQ). We note that when vector quantising the noise distribution is likely to be Gaussian rather than uniform, but argue that the uniform quantiser analysis gives us a reasonable starting target for quantiser performance.

The process used to train a VQ is:

Bits	Levels	Reconstruction Levels	Quant Error	σ^2	σ
1	2	$[-0.5, +0.5]$	$[-0.5, +0.5]$	0.0833	0.289
2	4	$[-0.75, -0.25, ...]$	$[-0.25, +0.25]$	0.0208	0.144
3	8	$[-0.875, -0.625...]$	$[-0.125, +0.125]$	0.0052	0.072
4	16	$[-0.9375, -0.8125...]$	$[-0.0625, +0.0625]$	0.0013	0.036
5	32	$[-0.96875, -0.90625, ...]$	$[-0.03125, +0.03125]$	0.0003	0.018

Table 1: Uniform Quantiser Examples. The quantisation error is for any input value x inside the outermost levels, i.e. $r_0 \leq x \leq r_{s-1}$.

Name	Input	Output
Autoencoder1	$\mathbf{b}$	$\hat{\mathbf{b}}$
Autoencoder2	$\mathbf{y}$	$\hat{\mathbf{y}}$
Autoencoder3	$\mathbf{b}$	$\hat{\mathbf{y}}$

Table 2: Autoencoder designs. $\mathbf{b}$ are mel spaced, smoothed $K = 20$ vectors, $\mathbf{y}$ are oversampled $K = 80$ linear spaced.

1. The autoencoder network is trained with Gaussian noise.
2. The network is then run in inference mode without quantisation noise and the latent bottleneck vectors saved to a file.
3. A VQ is trained on the latent vectors using *kmeans* that has a residual quantisation error σ^2.

1.1 Autoencoder 1 Results

Several autoencoders were tested, summarised in 1.1.

The autoencoder *autoencoder1.py* was trained with $\sigma^2 = 10^{-3}$ gaussian noise over 100 epochs, using the weighted linear loss function 4 and normalised $K = 20$ target vectors. The network was then run in inference mode, and the SD for various amounts of noise at the bottleneck measured (Table 1.1). Note that although the network was trained using a weighted linear loss function, inference performance was measured using the SD loss function.

With $\sigma^2 = 0$, the network was run again and the bottleneck (latent) vectors stored to a file.

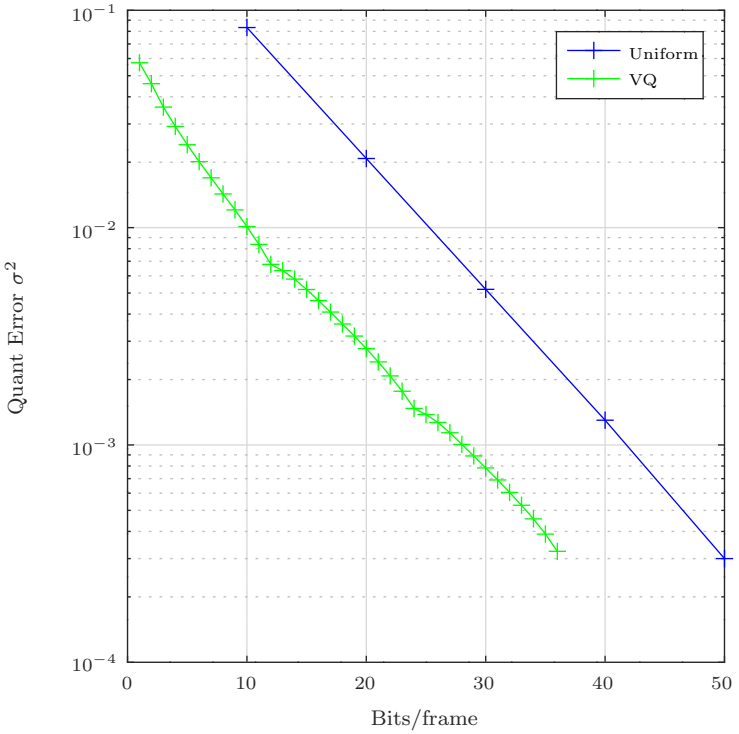
A PCA was performed on the latent vectors using GNU Octave. The PCA results showed significant linear correlation still exists in the bottleneck vectors, which suggests:

1. This autoencoder is non-optimal. An idea bottleneck dataset would have no measurable linear correlation between elements.
2. PCA can be a useful tool for measuring autoencoder performance (at least for linear dependencies).

σ^2	σ	SD dB^2	VQ bits	VQ stages	bits/s
0.000	0.000	1.89			
0.0003	0.018		36	3	1800
0.0010	0.031	2.57	28	3	1400
0.0015	0.038		24	2	1200
0.0030	0.055	3.93	20	2	1000
0.0100	0.100	8.77	10	1	500

Table 3: SD and multi-stage VQ bits for various levels of Gaussian bottleneck noise, using *nn4_cat1.pt* trained with weighted linear loss. The bit rate assumes the vectors are sent every 20ms (sufficient for communications quality speech).

Figure 1: Uniform and multi-stage VQ performance



3. With an improved network, a narrower bottleneck dimension d should be possible. This will improve multistage VQ performance, which is difficult when d is large as the residual error vectors tend towards independent Gaussians with no correlation the VQ can exploit.

A mesh plot was made of a segment of the latent vectors. This clearly showed significant time correlation between the same elements in adjacent vectors. A better autoencoder could exploit this time correlation, and jointly quantise adjacent vectors leading to a lower quantisation bit rate.

A multistage vector quantiser (each stage with 12 bits or 4096 vectors) was trained using the latent vectors as training material. The number of bits required to achieve the same variance as the intentionally injected noise is presented in 1. Figure 1 plots the performance of the uniform and vector quantisers. The VQ stage breakpoints at 12 and 24 bits can be observed. The VQ is around 15 bits/frame better than uniform quantisation for the same quantiser error σ^2. The slight downwards curve as the bits increase for each VQ stage is probably due to relatively small amount of training data (143,000 frames for up to $2^12 = 4096$ vectors per stage) resulting in some over-fitting.

As with the PCA results, the large slope of the first VQ stage (1-12 bits) suggests significant correlation in the bottleneck data that could alternatively be removed by an improved ML network, leaving independent Gaussians to be VQed.

Informal listening tests using headphones and laptop speakers (see *240118_wav* folder described in *README*):

1. In the male samples, there was a drop in quality due to the unquantised autoencoder with the buzzy artefact (typical of poor energy distribution across the pitch cycle) returning. This was more noticeable through headphones than laptop speakers. This suggests formant bandwidths are not encoded well. The network and VQ was trained with the target of minimising MSE in the smoothed $\mathbf{b}$ log mel-spaced vectors, rather than the high resolution $\mathbf{y}$ vectors that contain the bandwidth information.

2. The 24 bit VQ of the bottleneck vectors worked quite well, with no obvious quality drop compared to the unquantised autoencoder.

3. The unquantised autoencoder, VQ 12, and VQ 24 bit male samples were poorer than the reference Codec 2 3200 sample.

4. With 12 bit VQ the males suffered a further drop in quality, but through a laptop speaker the female sample sounded noisey but otherwise quite acceptable.

So overall results were not satisfactory for a practical quantisation system, however this appears to be due to autoencoder distortion rather than VQ. The hybrid ML/VQ scheme does appear viable, suggested we have trained a "well behaived" latent space that is amenable to quantisation. The use of "old school" externally trained VQ was likely non-optimal but removed the need to deal with

the many pitfalls of VQVAE [2]. The female results suggest low bit rates (12 bits/frame) may be possible if we can overcome the energy distribution issue. However the buzzy artefacts for males suggests running the quantisation scheme on the smoothed **b** vectors may not be the best choice - we should perhaps make the target the high resolution **y** vectors.

1.2 Autoencoder 2 and 3 Results

After some experimentation, improved results were obtained using *autoencoder3.py* by letting the network train for 200 rather than 50 epochs. This reduced the loss and produced acceptable quality speech with a $d = 10$ bottleneck. Quantising the bottleneck using a 24 bit VQ also produced acceptable speech. VQ distortion results similar to Table 1.1 were obtained, e.g. $\sigma^2 = 0.0015$ for 24 bits over the training database. Informal listening showed some audible distortion under quantisation through headphones, however the distortion was hard to detect through laptop speakers. Samples from inside the training database sounded better than samples from outside, however the VQ did not break down. The speech samples are available in the *240125_wav* folder.

The weighting factor γ was adjusted from 0.5 to 0.85, however it is unsure if this was useful. There was a slight bass artefact that was evident after dimensionality reduction, that may indicate a mismatch at low frequencies.

Finally, *autoencoder3.py* was tested with a $d = 10$ bottleneck but without quantisation. The output speech was quite good, with a slight bass artefact from samples outside the training database. This design is interesting as it removes the need for the intermediate **b** log mel-spaced samples, a staple of the ML vocoder world. Instead, the network trains it's own representation.

1.3 Conclusions and Further Work

Conclusions:

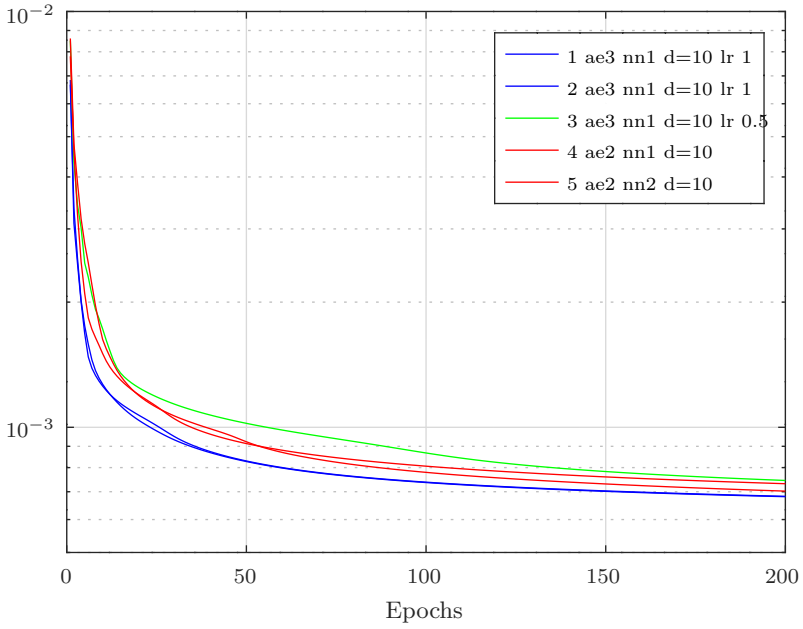
1. After some experimentation, successful $d = 10$ dimensionality reduction was obtained, and the bottleneck vectors quantised using a 24 bit VQ with acceptable results. Some robustness to samples outside the training data was demonstrated.

2. A scheme has been developed for combining ML with traditional VQ. While probably non-optimal compared to quantisation in the network, it is simple to train and implement and likely usable on small machines such as microcontrollers.

3. The VQ outperformed the baseline linear quantisation model, suggesting significant correlation still exists in the bottleneck vectors.

4. We have a useful objective measure σ^2 for selecting bit rate and designing quantisers.

Figure 2: Autoencoder 2 and 3 Loss versus Epochs. Note the significant drop from 50 to 200 epochs. The two blue lines represents two training runs of the same network, in this case they produce the same results.



5. A PCA showed significant linear dependencies still existed in the bottleneck vectors, and a mesh plot illustrated time dependencies. An improved network could remove these dependencies, leaving less information to quantise.

6. Training for extended periods and plotting loss against epochs was a useful (Figure 2). In this work, extending training from 50 to 200 epochs resulted in halving the loss and improved speech quality. This also suggests more training material is advisable. Some designs showed differing results with successive training runs; performing multiple training runs and comparing loss function plots is useful. Removing the mean from features helped in some autoencoder experiments, but is problematic to apply with weighted linear loss.

7. All three architectures in Table 1.1 could be considered for a practical quantisation system.

8. Buzzy male speech (poor energy distribution) is the classic breakdown artefact. A better network (or indeed baseline vocoder) that accurately modelled energy distribution would ensure buzzy speech is not possible.

Further Work:

1. Extend the network to consider blocks of samples in time, and use more sophisticated networks than simple FC layers.

2. Train the quantiser in the loop.

3. Demonstrate transparent dimensionality reduction.

4. Develop a quantiser that include the other vocoder parameter and the frame energy (the vectors quantised above were normalised).

5. Develop a fully quantised codec using these techniques and compare to existing Codec 2 modes.

6. Develop reduced memory networks that will fit in a few 10's of kbytes, for microcontroller implementation.

1.4 Weighted Linear Loss Function

From the manifold work [1], we have employed a weighted linear loss function:

$$L = \frac{1}{K} \sum_{k=0}^{K-1} |(x_k - \hat{x}_k) W(k)|^2 \tag{4}$$

where x_k are the linear spectral envelope samples. As this function operates in the linear domain, high energy parts of the spectrum (such as formants) will be preferentially optimised. However we also wish to pay some attention to

other parts of the spectrum. $W(k)$ is a weighting function that reduces the contribution of spectral peaks (while still maintaining a reasonable match to formant energy), and increases the contribution of spectral valleys. The input features to the ML system are expressed in dB:

$$\begin{aligned} X_k &= 20log_{10}x_k \\ x_k &= 10^{X_k/20} \end{aligned} \tag{5}$$

However to reduce dynamic range in the network X_k are scaled:

$$Y_k = log_{10}x_k = X_k/20 \tag{6}$$

The loss function used for training is:

$$L = \frac{1}{K}\sum_{k=0}^{K-1} |(10^{Y_k} - 10^{\hat{Y}_k})W(k)|^2 \tag{7}$$

Which can be shown to be the same as (4):

$$\begin{aligned} L &= \frac{1}{K}\sum_{k=0}^{K-1} |(10^{X_k/20} - 10^{\hat{X}_k/20})W(k)|^2 \\ &= \frac{1}{K}\sum_{k=0}^{K-1} |(x_k - \hat{x}_k)W(k)|^2 \end{aligned} \tag{8}$$

Unlike loss functions such as Spectral Distortion (SD):

$$SD = \frac{1}{K}\sum_{k=0}^{K-1} |(X_k - \hat{X}_k)|^2 \tag{9}$$

which can be expressed in dB^2, the units of the loss function L are difficult to interpret. It can be observed that L is the mean square error energy, with which has been re-distributed (shaped) by the weighting function $W(k)$.

This section WIP. Todos:

1. Include normalisation of energy $\sum_k |x_k|^2 = 1$, I think this makes $L_{max} \leq 1$.
2. Weighting function reduces peaks, which would make $L_{max} < 1$
3. See if we can find a reason for the factor of 400 currently used in code.

2 Unorganised Notes

1. Limit the amount of information through pre processing, e.g. compression, equalisation, dynamic range limiting. This would reduce the information in the log spectral vectors. Traditional DSP or ML? I feel this is where the biggest gains are.

2. Training a system that is robust to channel errors (or noise in an analog channel). How to get something close to MAP/or use LLRs and other info? A latent vector that can handle a lot of "noise".

3. Ensuring we get good energy distribution (formant bandwidths) under quantisation. Preserving bandwidth more important that frequency - can we include this weighting in loss function.

4. Need a meaningful way of interpreting the weighted loss as a comparable objective measure. It should converge to a similar result as MSE, but favour different parts of the spectrum - change the distribution of the noise. Perhaps just report regular SD in parallel.

5. L2 energy normalisation, expression, justification, reduce dynamic range during regression. Put energy "back in" later, as a separate feature

6. An alternative to injecting noise is quantising to a nearest reconstruction level. Good check. However what does gradient look like? Check again how additive noise is OK wrt to gradient.

7. To cross check substitute FVQ functions (coded so they don't blow up due to size)

8. A better network would perform more exotic transformations, organise a suitable transformation.

9. We can think of hybrid approaches, e.g. search latent space of output - analysis by synthesis I guess. Use that to obtain centroids etc. Fix NN while VQ search is in progress. I guess that leads us back to the VQ In the loop.

10. If we replace uniform with gaussian noise - can consider directly modulated symbols. Need to work out SNR.

References

[1] Unwrapping Codec 2 Data Manifolds. `https://github.com/drowe67/misc/manifold/manifold.pdf`.

[2] Fabian Mentzer, David Minnen, Eirikur Agustsson, and Michael Tschannen. Finite scalar quantization: Vq-vae made simple. *arXiv preprint arXiv:2309.15505*, 2023.