ML quantisation of Vocoder Features

January 14, 2024

The goal is to achieve improved quantisation using ML techniques:

1. Lower bit rate for equivalent distortion.
2. Increased robustness to channel errors that traditional modulation and FEC?

Expectations. There is quite a bit of 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 (including initial state information). 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). One could start with a ML transform of 4 x 10ms vectors, quantise that as an initial state, then quantise the residual errors from each 10ms frame.

Is running the VQ training loop (e.g. VQVAE) inside the ML system worth it? Several practical problems, such as the rate the VQ adapts compared to the rest of the network, managing 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 VQ could be simulated in the training loop using small amounts of AWGN injection to ensure the latent space is well behaved. Then, the VQ can be designed on the training data in the bottleneck. In general, use traditional DSP/quantisation where possible, leave ML to what it does best. Perhaps hybrid training, an epoch each.

Where to these fit in:

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 the VQ to minimise the weighted linear error (at $K = 20$ or via F at $K = 80$). Can this be performed using traditional VQ training? Would we use a low dim vector for the VQ?

3. 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".

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

1. Experiment 1: Goal is to demonstrate dimensionality reduction. Build an autoencoder for 1 vector. Try different architectures, e.g. FC and conv1D. 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.

4. Experiment 2: Build a VQVAE (VQ that trains in a ML network). Compare distortion to kmeans for a range of VQ sizes. Do they get the same performance?

5. Experiment 3: Goal is to reduce bit rate by considering longer time windows. Return to 1a (scalar linear quantisation) but attempt longer chunks of time (say 40 or 80 ms).

6. Experiment 4: Embed pitch in the latent space being quantised, as well as log spectral information.

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 $d = 10$ bottleneck was sufficient to maintain less than

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. $r_0 \leq x \leq r_{s-1}$.

$1dB^2$ distortion, indicating a useful dimension reduction. Good results (based on visual inspection) were also obtained using a weighted linear loss function 4.

However 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:

1. The autoencoder network is trained with Gaussian noise.

σ^2	σ	SD dB^2	VQ bits	VQ stages	bits/s
0.000	0.000	1.56			
0.0007	0.003		36	3	1800
0.0010	0.031	1.88	33	3	1650
0.0015	0.038		24	2	1200
0.0030	0.055	2.52	18	2	900
0.0100	0.100	4.80	10	1	450

Table 2: SD and multi-stage VQ bits for various levels of Gaussian bottleneck noise. The bit rate assumes the vectors are sent every 20ms (sufficient for communications quality speech).

2. The network is then run in inference mode without quantisation noise and the latent bottleneck vectors saved to a file.

3. An attempt is made to train a VQ using *kmeans* that has a residual quantisation error σ.

1.1 Results

The autoencoder was trained with $\sigma^2 = 1E-3$ 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 results 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:

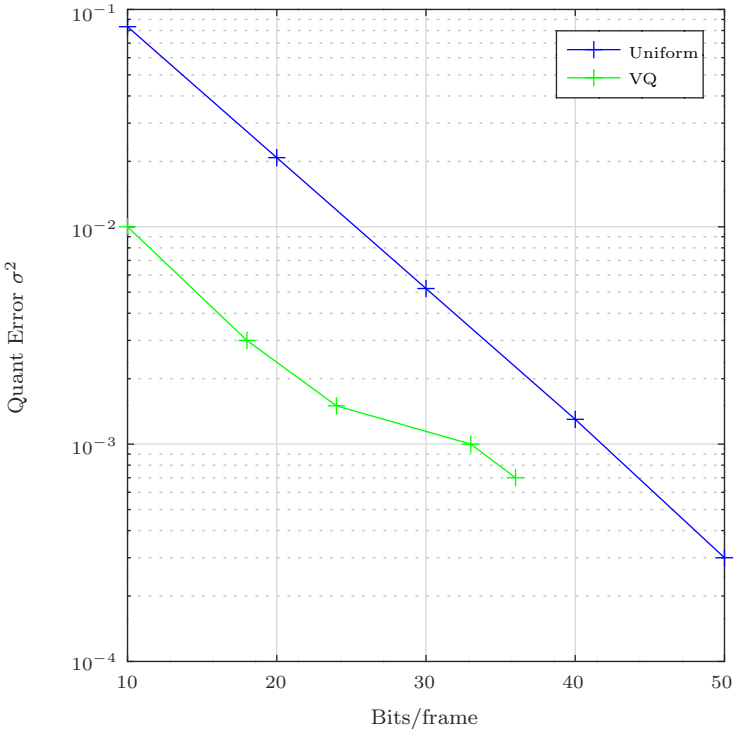
1. This autoencoder is non-optimal. An idea bottleneck dataset would have no measurable linear correlation between elements.

2. PCA can be a useful tool for measuring autoencoder performance (at least for linear dependencies).

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.

Figure 1: Uniform and VQ performance. The VQ has a significant advantage at low bit rates. The dip in the final VQ point is probably due to a marginal amount of training data (143k vectors) as we approach 12 bits per stage.



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. As with the PCA results, this suggests significant linear and non-linear correlation in the bottleneck data that could alternatively be removed by an improved ML network, leaving independent Gaussians to be VQed.

1.2 WIP Todos

1. 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.

2. This could be interpreted as 10x5 bits or a 50 bit quantiser

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

4. 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.

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

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

7. Useful to try a VQ at the bottleneck, kmeans and VQVAE trained, see how the information is organised. Will training the VQ based on the MSE of the latent space be meaningful? I guess it depends on how latent space is organised. So - very good idea to train a VQ against minimising MSE of this latent space.

8. 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.

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

1.3 Weighted Linear Loss Function

From the manifold work (TODO Ref), 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.