Opus Coarse Energy Predictor Notes (Round 3)

Jake Taylor (yupferris@gmail.com), 29 June 2021

I'm having trouble reconciling the coarse energy predictor implementation in the libopus source code and the 2D z-transform description in the paper[1]. At this point, I've gotten quite close, but am still having trouble with the last bits. And help/guidance would be appreciated!

I've simplified the source code (in unquant_coarse_energy in quant_bands.c in libopus 1.3.1[2]) to the following C (pseudo)code:

```
void unquant_coarse_energy(float *e, int bands) {
  float alpha = /* ... */;
  float beta = /* ... */;
  float prev = 0.0f;
  for (int b = 0; b < bands; b++) {
    float r = /* read from bitstream */;
    float q = alpha * e[i] + prev;
    e[i] = q + r;
    prev = prev + (1 - beta) * r;
  }
}
```

According to the paper, the 2D z-transform should be:

$$A(z_\ell, z_b) = (1 - \alpha z_\ell^{-1}) \cdot \frac{1 - z_b^{-1}}{1 - \beta z_b^{-1}}$$

First, to state what I think is obvious: the domain of this filter should be a 2D "energy plane" with the ℓ-dimension representing frames and the b-dimension representing bands, and the range should be the prediction (actual band energy minus $r[\ell, b]$, the residual). As a predictor, the filter must be causal. Finally, according to the code above, the energy is always 0 for $b < 0$ ($b \geq bands$, $\ell < 0$, and $\ell \geq frames$ are neither specified nor useful).

As outlined in the CELT blog post[3], this z-transform describes two (separable) cascaded predictors:

$$A(z_\ell, z_b) = P(z_\ell) Q(z_b)$$

The first is the ℓ-dimension predictor whose domain is indeed the 2D energy plane (albeit a single "row" of it corresponding to a particular band in isolation over multiple frames):

$$P(z_\ell) = 1 - \alpha z_\ell^{-1}$$

If we write the range explicitly:

$$P(z_\ell) = (1 - \alpha z_\ell^{-1}) E(z_\ell)$$

This corresponds to the following difference equation:

$$p[\ell] = e[\ell] - \alpha e[\ell - 1]$$

The range of this predictor is not the final prediction, but an intermediate p. Note that this equation includes the current band's energy $e[\ell]$, which is somewhat surprising for a predictor, as $e[\ell]$ will not be known until it can be derived after r is decoded in the decoder.

For each ℓ ("columns" of the energy plane), successive elements of p are then fed into the b-dimension predictor, whose range is the final prediction:

$$Q(z_b) = \frac{1 - z_b^{-1}}{1 - \beta z_b^{-1}}$$

The equivalent difference equation is:

$$q[b] = p[b] - p[b - 1] + \beta q[b - 1]$$

What remains now is to match these cascaded predictors with the C code, which is not trivial because it's not immediately apparent that the above derived difference equations are present in the code. There are likely several ways to do this; two high-level approaches stand out to me: either "lower" the expected z–transforms to difference equations and refactor until we get matching code, or "lift" the code to z-transforms and refactor until those match.

I find it easier to manipulate terms in the z-domain, so let's try the latter approach. We'll first describe $prev$ in terms of the known signals. Looking at the C code, we know the following:

$$q[\ell, b] = \alpha e[\ell - 1, b] + prev[\ell, b]$$

Since q is the output of the predictor, we can rewrite it in terms of e and r:

$$q[\ell, b] = e[\ell, b] - r[\ell, b]$$

This is also given by the C code. Substituting this in the previous equation yields:

$$prev[\ell, b] = e[\ell, b] - \alpha e[\ell - 1, b] - r[\ell, b]$$

Or, equivalently:

$$prev[\ell, b] = p[\ell, b] - r[\ell, b]$$

Trivially, the corresponding z-transform is:

$$Prev(z_\ell, z_b) = P(z_\ell, z_b) - R(z_\ell, z_b)$$

So, $prev$ apparently represents the *difference* between the output of the ℓ-dimension predictor p and the residual r. Interestingly, q, the expected range of the b-dimension predictor, has completely disappeared! This is odd, as without this term, it's going to be difficult to reach a z-transform with the correct range. However, ignoring this for now, we can still make some kind of progress.

The difference equation governing $prev$ is the following (again, from the C code):

$$prev[\ell, b + 1] = prev[\ell, b] + (1 - \beta) r[\ell, b]$$

[1]https://arxiv.org/abs/1602.04845
[2]https://opus-codec.org/release/stable/2019/04/12/libopus-1_3_1.html
[3]https://jmvalin.dreamwidth.org/12000.html

Note that the output of the difference equation is indexed with $b+1$ because it represents the value of $prev$ *after* it has been updated for this loop iteration. This is actually an important distinction, because the code that modifies $prev$ uses the $r[\ell, b]$, i.e. the *current* residual, and these two signals should have the correct phase with respect to one another. We can also express this as:

$$prev[\ell, b] = prev[\ell, b-1] + (1-\beta)r[\ell, b-1]$$

These two definitions are, of course, equivalent. The z-transform of this difference equation is:

$$Prev(z_\ell, z_b) = z_b^{-1} Prev(z_\ell, z_b) + (1-\beta)z_b^{-1} R(z_\ell, z_b)$$

The final step involves substituting occurrences of $prev$ with equivalent expressions in terms of P and R:

$$P(z_\ell, z_b) - R(z_\ell, z_b) = z_b^{-1}(P(z_\ell, z_b) - R(z_\ell, z_b)) + (1-\beta)z_b^{-1} R(z_\ell, z_b)$$

This can be simplified to:

$$\frac{R(z_\ell, z_b)}{P(z_\ell, z_b)} = \frac{1 - z_b^{-1}}{1 - \beta z_b^{-1}}$$

While this filter has *almost* the expected definition with the correct domain p, its range is r, *not* q! This is somewhat unsurprising, as the q terms cancelled out of our equations above. However, I find it really strange that the output of the predictor is actually the residual, and not the prediction itself. One would think that, because the residual is not yet known to the decoder until dequantization, the predictor would represent the resulting energy minus that residual. I suppose I could be happy accepting that, but it would still be nice to get a few more pairs of eyes on this to make sure I haven't made any mistakes, and perhaps to provide additional insight on why the predictor is factored this way.