Lab 2: Source Coding: Turning Signals into Bits

Report Due: 21:00, 4/12, 2019

Overview

Source coding is a mapping from (a sequence of) symbols from an information source to a sequence of alphabet symbols (usually bits) such that the source symbols can be exactly recovered from the binary bits (lossless source coding) or recovered within some distortion (lossy source coding). This is the concept behind data compression.

Experiments

In this section, we want to experiment with audio signals and to observe the output signal after processing. The first part of Lab 2 is to use short-time Fourier transform (STFT) to determine the sinusoidal frequency content of local sections of a signal as it changes over time. The second part of Lab 2 is try to implement Huffman code, which is commonly used for lossless data compression.

1. **Short-time Fourier transform:** Using the following sample $x(t)$ that is composed of a set of four sinusoidal waveforms joined together in sequence. Each waveform is only composed of one of four frequencies {10, 25, 50, 100 Hz}. The definition of $x(t)$ is:

$$x(t) = \begin{cases} \cos(2\pi \times 10t) & , 0 \leq t < 1\text{s} \\ \cos(2\pi \times 25t) & , 1 \leq t < 2\text{s} \\ \cos(2\pi \times 50t) & , 2 \leq t < 3\text{s} \\ \cos(2\pi \times 100t) & , 3 \leq t < 4\text{s} \\ 0 & , \text{otherwise.} \end{cases}$$

   Then it is sampled at 400 Hz.

   (a) Using a **rectangular window** with window size is 1s, capture $x(t)$ in $0 \leq t < 1s$. We call this section $x_1(t)$. Plot $x_1(t)$ with respect to the time t in seconds. Plot the magnitude of **CTFT** of $x_1(t)$, with the horizontal axis is the frequency f in the range of $[-f_s/2, f_s/2]$. Please **add axis labels**, and describe what you observe.

   (b) Using the same window, slide the window and capture $x(t)$ in $2.5 \leq t < 3.5s$. We call this section $x_2(t)$. Plot $x_2(t)$ with respect to the time t in seconds. Plot the magnitude of **CTFT** of $x_2(t)$, with the horizontal axis is the frequency f in the range of $[-f_s/2, f_s/2]$. Please **add axis labels**, and describe what you observe.

   (c) Now your job is to write your own short-time Fourier transform function to implement time-frequency analysis over the $x(t)$. Your own short-time Fourier transform function should have the following function arguments:

```
[S, f, t] = STFT(x, window, Noverlap, Nfft, fs)
```

The inputs/outputs of this function is similar to the `Matlab` function `spectrogram()`. You can verify your `STFT()` with `spectrogram()`, but there are some differences.

- Inputs:
  - `x`: A column vector of input time signal.
  - `window`: Specified as a column vector. Use `window` to divide the input signal `x` into sections `xw`s.
  - `Noverlap`: Number of overlap samples.
  - `Nfft`: Number of DFT points, usually larger than (i.e., zero padding is performed) or equal to the length of `window`. (Example of zero-padding: `A = [1 2 3]`, if we want `A` to be a length-5 vector, we can pad 2 zeros after `A`, i.e. `A_zero_padding = [1 2 3 0 0]`)
  - `fs`: Sampling frequency.
- Outputs:
  - `S`: Short-time Fourier transform, returned as a matrix, with time increasing across the column of `S` and frequency increasing down the rows, starting from $-fs/2$.
  - `f`: Cyclical frequencies (Hz), returned as a vector. `f` has a length equal to the number of rows of `S`.
  - `t`: Instant of time, returned as a vector. $t(i)$ = middle-point of i-th `xw` section.

(d) Adjust the window size, and `Noverlap` to see how the `S` changes. You can use `Matlab` functions, `surf` and `colorbar`, to show `S`. Please **add axis labels** and describe what you observe.

(e) Perform short-time Fourier transform over `handel.ogg` and processed `handel.ogg` in Lab1 (i.e., hard-limiting, squaring, modulation, quantization). Show the spectrograms and describe what you observe.

2. **Huffman coding:**

(a) (Handwriting) Construct a Huffman tree by using these symbol. Everyone's method of implementing Huffman coding may be different, so please write on your calculation process.

(b) (Handwriting) Following (a), encode the sequence of symbols using the Huffman tree constructed in (a):

{d, a, b, h, c}

(c) (Handwriting) Encode the sequence of symbols in (b) using **PCM**.

(d) Compare the data size between (b) and (d).

(e) In Lab1, you have written a `Matlab` function `quantizer_L_level`. Here you should write a `Matlab` PCM encoder function with the following auguments

```
y = pcm_enc(x, numBits)
```

Table 1: A table of symbols and their probability

Symbol	Prob.
a	0.3
b	0.2
c	0.2
d	0.1
e	0.05
f	0.05
g	0.05
h	0.05

where `x` is output signal of `quantizer_L_level`, `numBits` is the number of bits of each codeword, and `y` is PCM bit stream.

(f) Write a `Matlab` function to construct Huffman code dictionary with following arguments:

```
dict = huffman_dict(symbols, p)
```

where `symbols` are distinct signal values of source, `p` is a probability vector whose kth element is the probability of kth symbol, and `dict` is a two-column cell array in which the first column lists the distinct signal values from `symbols` and the second column lists the corresponding Huffman codewords. The form of dict is same as `Matlab` function `huffmandict`.

(g) Write a Huffman encoder function which has the following arguments:

```
y = huffman_enc(x, dict);
```

where `x` is output signal of `quantizer_L_level`.

(h) Write a Huffman decoder function which has the following arguments:

```
x_dec = huffman_dec(y, dict);
```

where `y` is a bit stream, and `x_dec` represents the decoded x.

3. **Converting Signals into Bits:** We want you to convert analog signal to digital signal via the techniques containing **sampling & quantization & mapping**. Please turn `handel.ogg` into bits.

(a) After sampling (i.e., `audioread`) and quantization (i.e., `quantizer_L_level`), Huffman coding needs the probability distributions of samples (symbols). Count the number of occurrences of samples, and convert it into probabilities.

(b) Following (a), map samples to bits using Huffman coding.

(c) After sampling and quantization, map samples to bits using PCM.

(d) Compare the data size of (b) and (c).

(e) Please decode the digital bits in (b) and (c) to digital signals. Compare two recovered `handel.ogg`.

(f) Encode the processed `handel.ogg` (i.e., hard-limiting, squaring, modulation) using Huffman coding and PCM, and compare the data size.

4. **The Average Codeword Length of Huffman Codes:** In this problem, we will study the average codeword length of Huffman codes via Monte-Carlo simulation. We will consider the symbols $\mathscr{S} = \{\mathrm{a}, \mathrm{b}, \ldots, \mathrm{h}\}$ in Table 1.

   (a) Calculate the entropy of $\mathscr{S}$, i.e. $H(\mathscr{S})$.

   (b) Find the average codeword length $\overline{L}$ of the Huffman code.

   (c) Generate a sequence of $n = 100$ symbols according to the probability in Table 1. Find the length of encoded binary data (in bits) after Huffman encoding.

   (d) Repeat the experiment in Problem 4c multiple times (say $R = 1000$ times). Each experiment randomly generates a sequence of symbols. Assume that the length of the encoded binary data for the r-th experiment is $L_n^{(r)}$. Compute the average length of the binary data according to

$$\overline{L}_n = \frac{1}{R} \sum_{r=1}^{R} L_n^{(r)}. \tag{1}$$

   (e) Compare the quantity $\overline{L}_n/n$ in Problem 4d with the entropy $H(\mathscr{S})$ and the average codeword length $\overline{L}$ in Problems 4a and 4b. Comment on your results.

   (f) Repeat Problem 4e with $n = 1000$ and $R = 10000$.

Lab Report

There is no format requirements for your lab report. In the report, you should address the results of the exercises mentioned above. You should also include your simulation program in the appendix of the report. Include whatever discussions about the new findings during the lab exercise, or the problems encountered and how are those solved. Do not limit yourself to the exercises specified here. You are highly encouraged to play around with your simulation program on self-initiated extra lab exercises/discussions. For example, you can record your own voice or search for sound clips with no copyright at Freesound.org (`https://freesound.org/`).