哈爾濱工業大學

实验报告

实 验（一）

题　　目　DPCM 编解码实验

专　　业　人工智能

学　　号

班　　级

学　　生

指 导 教 师　郑铁然

实 验 地 点　格物楼 213

实 验 日 期　2024.4.7

计算机科学与技术学院

一、 8 比特 DPCM 编解码算法

1.1 简述算法内容

DPCM（Differential Pulse Code Modulation，差分脉冲编码调制）是一种数字信号编码技术，主要用于无损压缩和数据传输领域。其基本原理是对信号的差分进行编码和解码，而不是直接对信号进行采样和编码。本实验基于**量化因子法**主要实现了以下部分：

1.1.1 DPCM 编码器（DPCM_encode_quantized）:

如图 1 所示，编码器利用比特数(num_bits)确定量化级别，即quantization _levels $= 2^{num\_bits-1}$。将原始信号的第一个样本直接添加到编码列表中作为初始值。而后遍历原始信号(signal)，对每个样本进行如下处理：

a) 首先计算当前样本与预测值(prediction)之间的差异(difference)，然后通过调用 quantizer 函数将差异值进行量化，得到量化后的差异值(quantized_difference)，将其添加到编码信号列表中。

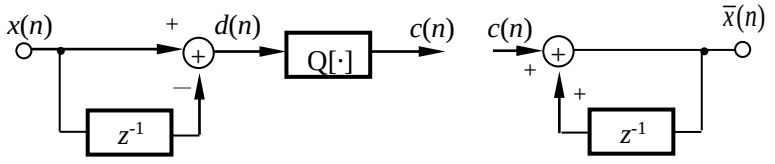
b) 随后，更新预测值，以便下一个样本的预测。更新方式为将差异值减去量化级别数量后乘以量化因子(factor)，然后加到预测值上。

最后，返回原始信号的第一个样本值和编码后的信号列表，用于后续的解码过程。

1.1.2 DPCM 解码器（DPCM_decode_quantized）:

解码器同样根据量化比特数计算量化级别 (quantization_levels)。接着，将初始样本值赋值给数组的第一个元素(data[0])，因为第一个样本的值在编码过程中保持不变。随后，通过遍历编码后的信号(encoded_signal)，对每个样本进行解码操作。

解码过程其实就是编码的逆向过程，将编码信号减去量化级别数量后乘以量化因子(factor)，然后加到前一个样本的值(data[i-1])上即可得到当前样本的值(data[i])。



编码器　　解码器

图 1 DPCM算法流程

1.1.3 量化器（quantizer）:

本实验采用**量化因子法**进行量化，量化器将给定的差异值(x)根据指定的量化比特数(num_bits)和量化因子(factor)进行量化处理。总结为如下公式：

$$c(n)=\begin{cases} ql-1 & d(n)>ql\times a \\ -ql & d(n)<-ql\times a \\ i & (i-1)a<d(n)\le ia, -ql\le i\le ql-1 \end{cases}$$

当$a=1$时，**量化因子法**实质上就等于直接量化法。

1.1.4 压缩和解压函数:

在压缩函数中，将奇数位置的值（高八位）与偶数位置的值（低八位）组合成一个压缩数据，并存储到数据类型为 np.int16 的压缩数据数组中。

而解压函数则逆向操作，遍历压缩数据，将每个压缩数据进行按位与操作，从中提取高位和低位，然后将它们分别放入还原后的数组中。

这样，这两个函数实现了对编码后的信号进行有效的压缩和解压缩，以减少数据存储和传输所需的空间。

1.1.5 信噪比计算函数（calculate_SNR）:

接收原始信号和解码后的信号作为输入。计算原始信号和解码后信号之间的信噪比，用于评估压缩算法的性能。

$$SNR = 10 * \log_{10}\left\{\frac{\sum_{n=0}^{M}(x(n))^2}{\sum_{n=0}^{M}(\bar{x}(n)-x(n))^2}\right\}$$

1.2 解码信号的信噪比

结果如下表所示，可以发现量化因子并不是越大越好，也不是越小越好，原因可能是：

a) 过大的量化因子会限制了编码的动态范围，使得编码后的信号无法准确反映原始信号的变化；

b) 过小的量化因子可能会导致量化误差的幅度过小，在量化后的信号中占据较少的比特数，可能会增加量化误差对信号质量的影响，使得编码后的信号失真程度较高。

表格 1 解码信号信噪比（8-bit）

文件名	Factor=600	Factor=300	Factor=100	Factor=50
1.wav	17.233	23.522	31.260	20.917
10.wav	15.758	20.902	29.606	14.831
2.wav	24.470	30.581	37.601	13.099
3.wav	10.837	16.002	25.507	31.602
4.wav	16.269	22.685	32.096	24.550
5.wav	18.178	25.085	32.966	19.986
6.wav	21.312	27.327	36.913	35.041
7.wav	19.685	25.692	35.031	37.092
8.wav	16.161	22.036	31.067	23.684
9.wav	17.767	24.231	32.776	16.410

二、4 比特 DPCM 编解码算法

2.1 你所采用的量化因子

$Factor = 800,600,300,100$

2.2 拷贝你的算法，加上适当的注释说明

算法基本和第一节一致，除了压缩时采用的数据类型变为 int8，存放改为 8 位位操作，完整代码见附件 dpcm_with_4bit.py。

```
1.  def quantizer(x, num_bits, factor):
2.      """ 量化器 """
3.      quantization_levels = 2 ** (num_bits - 1)
4.      high = quantization_levels - 1
5.      low = -1 * quantization_levels
6.      bias = quantization_levels
7.      if x > high * factor:
8.          return high + bias                  # 加上 bias，变为非负值，方便压缩
9.      elif x < low * factor:
10.         return low + bias
11.     else:
12.         i = int(np.ceil(x / factor))        # 向上取整
13.         return i + bias
14.
15. def compress(encoded_signal):
16.     """ 压缩 """
17.     num = len(encoded_signal)
18.     compressed_data = np.zeros(int(num/2), dtype=np.int8)
19.     for i in range(0, num-1, 2):
20.         # 将奇数位置的值放在高四位，偶数位置的值放在低四位
21.         compressed_data[int(i/2)] = (np.int8(encoded_signal[i]) << 4) + (np.int8
    (encoded_signal[i + 1]))
22.     return compressed_data
23.
24. def decompress(compressed_data):
25.     """ 解压 """
26.     num_compressed = len(compressed_data)
27.     x = np.zeros(num_compressed * 2, dtype=np.int8)     # x 用来存还原后的量化误差
28.     for i in range(num_compressed):
29.         # 解压缩，将高四位和低四位分别放入还原后的数组中
30.         x[2 * i + 1] = compressed_data[i] & np.int8(int('00001111', 2))
31.         x[2 * i] = (compressed_data[i] >> 4) & np.int8(int('00001111', 2))
32.     return x
33. def DPCM_encode_quantized(signal, num_bits, factor):
34.     """ DPCM 编码器(量化因子为 factor) """
35.     quantization_levels = 2 ** (num_bits - 1)
36.     encoded_signal = [signal[0]]  # 初始化编码信号列表，将第一个样本直接添加到列表中
```

```
    prediction = signal[0]          # 初始化预测值为第一个样本值

    for sample in signal[1:]:       # 对信号中的每个样本进行编码
        difference = sample - prediction    # 计算当前样本与预测值之间的差异
        quantized_difference = quantizer(difference, num_bits, factor) # 量化
        encoded_signal.append(quantized_difference) # 将量化后的差异添加到编码列表
        prediction += (quantized_difference - quantization_levels) * factor#更新

    return signal[0], encoded_signal

def DPCM_decode_quantized(begin, encoded_signal, num_bits, factor):
    """ DPCM 解码器(量化因子为 factor) """
    quantization_levels = 2 ** (num_bits - 1)
    data = np.zeros(len(encoded_signal) + 1, dtype=np.int16)
    data[0] = begin.item()          # 第一个点一直保持不变
    for i in range(1, len(encoded_signal)):
        data[i] = data[i - 1] + (encoded_signal[i] - quantization_levels) * fact
or                                  # 预测下一个点
    return data
```

2.3 解码信号的信噪比

结果如下表所示，对比 8-bit 量化来说，4-bit 所需的量化因子更大，原因是 4-bit 量化级别的数量相对较少。如果量化因子过小，量化级别之间的间距可能会太小，导致量化后的信号的精度不足以准确表示原始信号的变化，从而造成较大的量化误差，信号失真严重。

因此，4 比特量化需要相对较大的量化因子，以确保量化后的信号能够有效地表示原始信号的动态范围。

表格 2　解码信号信噪比（4-bit）

文件名	Factor=800	Factor=600	Factor=300	Factor=100
1.wav	13.672	14.404	11.293	-0.550
10.wav	10.464	8.132	2.728	-3.603
2.wav	11.414	7.090	0.747	-6.773
3.wav	8.899	10.837	15.995	11.755
4.wav	13.137	14.257	9.399	-1.362
5.wav	13.837	13.911	10.542	1.506
6.wav	18.953	19.708	8.779	-2.949
7.wav	17.092	17.831	9.530	-0.902
8.wav	13.061	12.391	4.923	-2.756
9.wav	12.118	9.892	3.393	-3.154

三、改进策略

3.1你提出了什么样的改进策略，效果如何

3.1.1 下采样压缩:

在音频处理中，下采样压缩通常是通过减少音频信号的采样率来实现的。例如，将音频信号的采样率从原始的 CD 质量（44.1 kHz）降低到更低的采样率（如 22.05 kHz 或 11.025 kHz），从而减少了音频文件的大小。这样做会导致一定程度的音质损失，特别是高频信号的丢失，但可以在一定程度上节省存储空间。

本实验实现了对输入的原始信号数据进行平均下采样，通过对改变下采样的因子，对相邻信号做平均池化，从而将原始信号的采样率降低为原来的$1/n$，代码如下所示：

```
def downsample(data, factor):
    """ 平均下采样 """
    downsampled_data = []
    for i in range(0, len(data), factor):
        if i + 1 >= len(data):
            downsampled_data.append(np.short(data[i]))
            break
        downsampled_data.append(np.short(np.mean(data[i:i+factor])))
    return downsampled_data
```

经过下采样压缩后，还需在解码时，按照相同的比例因子进行还原，例如，当比例因子为 2 时，可以通过 upsampled_data = [decoded_data[i // 2] for i in range(len(decoded_data) * 2)]来得到与原本信号长度相等的解码信号。

实验结果可以看到，相较于应用下采样之前，压缩文件大小能够缩小到原来的$1/n$。

9_4bit.dpc	2024/4/8 15:06	DPC 文件	24 KB
9_8bit.dpc	2024/4/8 14:57	DPC 文件	47 KB
9_downsample_4bit.dpc	2024/4/8 13:57	DPC 文件	12 KB
9_log_4bit.dpc	2024/4/8 13:54	DPC 文件	24 KB
10_4bit.dpc	2024/4/8 15:06	DPC 文件	74 KB
10_8bit.dpc	2024/4/8 14:57	DPC 文件	148 KB
10_downsample_4bit.dpc	2024/4/8 13:57	DPC 文件	37 KB
10_log_4bit.dpc	2024/4/8 13:54	DPC 文件	74 KB

图 2 下采样压缩效果（$n=2$）

3.1.2 对数量化:

对差分信号进行**对数量化**处理。首先，根据所需的比特数计算量化级别的数量。然后，将差分信号取绝对值后取对数，将对数值四舍五入并限制在合理的量化级别范围内，最后根据差分信号的正负确定量化后的值。这样做的目的是根据信号的幅度来量化信号，使得在信号的较小变化范围内，量化精度更高。

设对数的底为a，在 4-bit 精度下进行量化的数学表示如下：

$$c(n) = \log_a|d(n)|, c(n) = min(c(n), 7) + \text{sgn}\big(d(n)\big) * 8$$

$$\bar{x}(n) = \bar{x}(n-1) - (-1)^{\text{sgn}(c(n)-8)} * a^{c(n)\&7}$$

其中sgn(·)函数表示当大于 0 时为 1，小于等于 0 时为 0。具体代码如下：

```python
def log_quantizer(difference, num_bits):
    """ 对数量化器 """
    quantize_levels = 2 ** (num_bits - 1)
    # c(n) = log(|d(n)|)
    # c(n) = min(c(n), quantize_levels - 1) + sgn(-d(n)) * quantize_levels
    if difference == 0:
        difference = 1
    log_difference = math.log(abs(difference))
    quantized_difference = min(max(round(log_difference), 0), quantize_levels-1)
    # (-quantize_levels, quantize_levels - 1)
    quantized_difference = quantized_difference + quantize_levels if difference > 0 else quantized_difference

    return quantized_difference

def DPCM_encode_log_quantized(signal, num_bits):
    """DPCM 编码器"""
    quantize_levels = 2 ** (num_bits - 1)
    encoded_signal = [signal[0]]
    prediction = signal[0]
    for sample in signal[1:]:
        difference = sample - prediction
        quantized_difference = log_quantizer(difference, num_bits)
        encoded_signal.append(quantized_difference)
        # x(n) = x(n-1) + ((-1)^sgn(c(n)- quantize_levels)) * (exp(c(n) & 7))
        sgn = 1 if quantized_difference >= quantize_levels else -1
        prediction += sgn * round(math.exp(quantized_difference & 7))

    return signal[0], encoded_signal

def DPCM_decode_log_quantized(begin, encoded_signal, num_bits):
    """DPCM 解码器"""
    # print(encoded_signal)
    # print(sgn)
    quantize_levels = 2 ** (num_bits - 1)
    data = np.zeros(len(encoded_signal) + 1, dtype=np.short)
    data[0] = begin.item()
    for i in range(1, len(encoded_signal)):
        sgn = 1 if encoded_signal[i] >= quantize_levels else -1
        data[i] = data[i - 1] + sgn * round(math.exp(encoded_signal[i] & (quantize_levels - 1)))
        # print(data[i])
    return data
```

实验结果如下，可以发现压缩效果波动很大，怀疑是没有量化，信号的动态范围较低，从而对信号的变化比较剧烈的序列压缩效果较差，可以考虑结合对数量化和量化因子法结合使用，但需要对数据的范围、符号有较为精细的设计。

表格 3　对数量化解码信号信噪比（4-bit）

文件名	对数量化信噪比
1.wav	3.116
10.wav	-1.554
2.wav	-3.690
3.wav	18.196
4.wav	1.647
5.wav	4.863
6.wav	0.368
7.wav	2.234
8.wav	-0.536
9.wav	-1.433

四、 简述你对量化误差的理解

4.1 什么是量化误差？

量化误差指的是在模拟信号经过离散化处理后，由于离散级别的限制而产生的近似误差。这种误差主要是由于连续信号的模拟数值被近似为量化器中的离散级别所引起的。其大小受两个主要因素影响：

1) **量化间隔（或量化步长）**：量化间隔是指相邻两个离散级别之间的距离。间隔越小，量化器的精度越高，因此引入的量化误差就越小；反之，间隔越大，引入的误差也越大。

2) **原始信号的幅度变化**：当原始信号的幅度变化范围较大时，部分信号可能会被量化到离散级别的边缘，导致更大的近似误差。

量化误差通常被视为系统中的一种噪声，它与信号的信噪比密切相关。较小的量化误差意味着更准确的信号重建，从而提高了信号的质量和信噪比。

4.2 为什么编码器中会有一个解码器

在 DPCM（差分脉冲编码调制）系统中，编码器内部包含解码器的原因是 DPCM 编码器内部存在一个**反馈环路结构**，编码过程依赖于先前重构样本的值来预测当前样本的值，并计算预测误差，随后对预测误差进行编码。

输入样本经过预测和减法器得到预测误差，然后经过量化和编码输出。同时，量化后的预测误差通过解码器解码和重构，以恢复原始的预测误差，再加到预测值上，得到当前重构样本，作为下一个样本的预测参考。

通过解码器的这一过程，DPCM 系统实现了对信号的连续预测和重建，从而使得编码器可以基于先前的重构样本进行预测，进而实现对信号的高效编码和压缩。

五、 总结

5.1 请总结本次实验的收获

理解了 DPCM 编解码算法的基本原理和实现过程。DPCM 编解码算法是一种音频压缩算法，通过预测和量化差值的方式来减少音频数据的存储空间。

学会了使用编程语言进行音频处理，掌握了编码、量化、压缩、解压、解码等关键步骤的实现方法，了解了多种量化方式如直接量化、量化因子、对数量化等，并通过实际调试完成 DPCM 算法的实现。

了解了信噪比的计算方法和评估音频解码效果的方式。通过计算信噪比，我们可以客观地评估解码后的音频数据与原始音频数据的准确度。

了解了降采样在音频处理中的应用。通过降采样可以减少数据量和计算复杂度，提高算法的效率和性能。

5.2 请给出对本次实验内容的建议

建议提供更多的参考资料，同时能在实验报告中介绍一些相关前言的理论技术的应用。