# 哈爾濱工業大學

# 实验报告

## 实 验（二）

题　　目 DPCM 编解码实验

专　　业 人工智能

学　　号 1190202107

班　　级 1903602

学　　生 姚舜宇

指 导 教 师 郑铁然

实 验 地 点 格物 213

实 验 日 期 2021.12.11

## 计算机科学与技术学院

# 一、 8 比特 DPCM 编解码算法

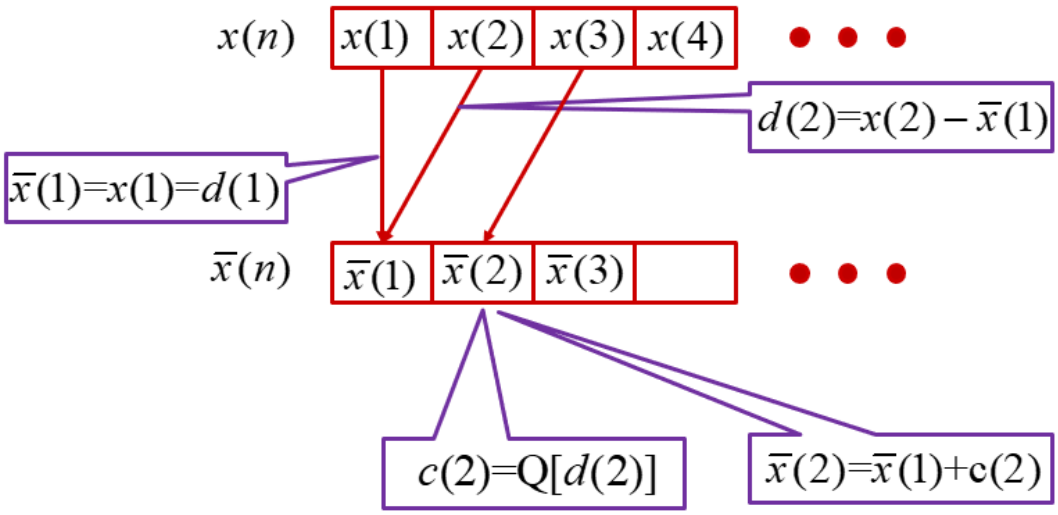
## 1.1 简述算法内容

1.读取.wav 文件，构造数据

两字节表示一个取样值，使用 short 类型，将十六进制字符串转为 short 型列表存储。

2.对原始数据进行 dpcm 压缩

按下图规律进行计算




用通项公式可以表示为：

$$d[i]=x[i]-\bar{x}[i-1]*u$$

$$c[i]=Q(d[i],a)$$

$$\bar{x}[i]=\bar{x}[i-1]*u+(c[i]-128)*a$$

其中$Q(d,a)=\begin{cases}255\ if\ d>127*a\\0\ if\ d<-128*a\\j+128\ if\ (j-1)*a<d\le j*a\end{cases}$，u 为权重，改进方法中会详细解释。

最后返回 c。

3.对压缩后的数据进行封装

使用二进制存储，每个字节存储一个取样，用列表保存。

4.将封装后的数据写入.dpc 文件

直接以二进制形式写入文件即可。

5.从.dpc 文件读取数据并进行解码

先从文件中以二进制形式读取，然后进行解码。解码方法为：

$$\bar{x}[i+1]=\bar{x}[i]*u+(c[i+1]-128)*a$$

6.将解码出的数据写入.pcm 文件

设置波形参数并以 wave 形式写入即可。

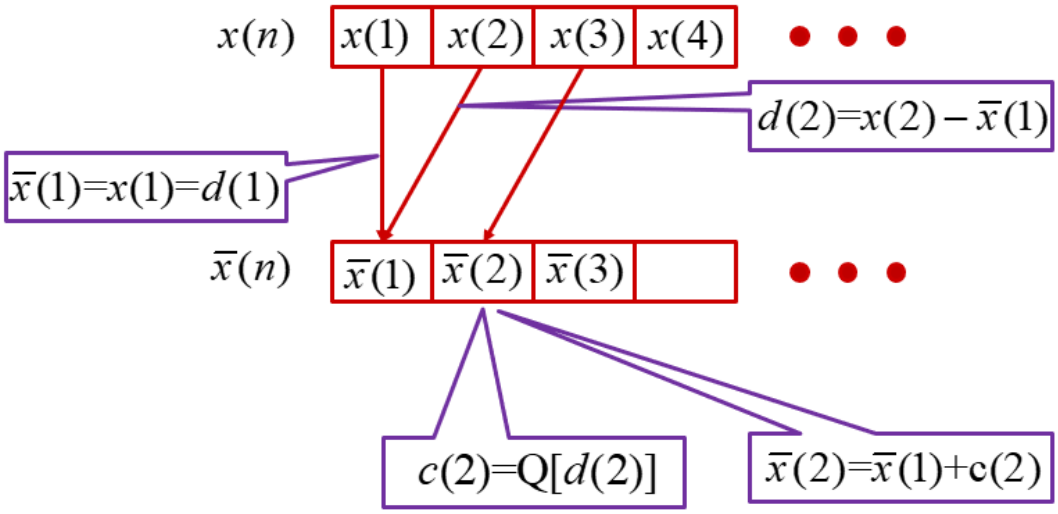
## 1.2 解码信号的信噪比

| | |
|---|---|
| 1.wav | 32.60 |
| 2.wav | 42.10 |
| 3.wav | 24.32 |
| 4.wav | 31.10 |
| 5.wav | 33.21 |
| 6.wav | 35.68 |
| 7.wav | 33.89 |
| 8.wav | 30.01 |
| 9.wav | 32.71 |
| 10.wav | 29.03 |

# 二、4 比特 DPCM 编解码算法

## 2.1 简述算法内容，给出所采用的编码参数（如所采用的量化因子等）

基本情况同 8 比特 DPCM 编解码算法。下面简述区别。
在对原始数据进行 dpcm 压缩的部分，同样按下图规律进行计算




用通项公式可以表示为：

$$d[i]=x[i]-\bar{x}[i-1]*u$$

$$c[i]=Q(d[i],a)$$

$$\bar{x}[i]=\bar{x}[i-1]*u+(c[i]-7.5)*a$$

其中$Q(d,a)=\begin{cases}15\ if\ d>7.5*a\\0\ if\ d<-7.5*a\\\dfrac{2j-1}{2}+7.5\ if\ (j-1)*a<d\le j*a\end{cases}$，u 为权重，改进方法中会详细解释。

最后返回 c。

在从.dpc 文件读取数据并进行解码部分，先从文件中以二进制形式读取，然后进行解码。解码方法为：

$$\bar{x}[i+1]=\bar{x}[i]*u+(c[i+1]-7.5)*a$$

## 2.2 拷贝你的算法，加上适当的注释说明

```
1.  import struct
2.  import wave
3.  import numpy as np
4.
5.  a = 890
6.  u = 0.91
7.  params = ()
8.  # is8or4 = '8'  # a=115, u=0.85
9.  is8or4 = '4'  # a=890, u=0.91
10.
11.
12. def readfile(filepath):
13.     """
14.     从.wav 文件中读入数据
15.     :param filepath: 文件路径
16.     :return: 数据
17.     """
18.     f = wave.open(filepath, 'rb')
19.     global params
20.     params = f.getparams()
21.     str_data = f.readframes(params[3])
22.     f.close()
23.     # 两字节表示一个取样值，使用 short 类型
24.     wave_data = np.fromstring(str_data, dtype=np.short)
25.     return wave_data
26.
27.
28. def dpcm(wave_data, a):
29.     """
30.     dpcm 编码
31.     :param wave_data: 原始数据
32.     :param a: 量化因子
33.     :return: 编码后的数据
34.     """
35.     x = wave_data
36.     length = len(x)
37.     x_hat = list(range(length))
38.     d = list(range(length))
39.     c = list(range(length))
40.     x_hat[0] = x[0]
41.     d[0] = x[0]
42.     for i in range(1, length):
43.         d[i] = x[i] - x_hat[i - 1] * u
44.         if is8or4 == '8':
45.             c[i] = _8quantification(d[i], a)
46.             x_hat[i] = x_hat[i - 1] * u + (c[i] - 128) * a
47.         else:
48.             c[i] = _4quantification(d[i], a)
49.             x_hat[i] = x_hat[i - 1] * u + (c[i] - 7.5) * a
50.     return c
51.
52.
53. def _8quantification(d, a):
54.     """
55.     8bit 直接量化法
```

```
    :param d: 差分值
    :return: 量化后的值
    """
    fc = 127
    if d > 127 * a:
        fc = 127
    elif d < -128 * a:
        fc = -128
    else:
        for j in range(-128, 127):
            if (j - 1) * a < d <= j * a:
                fc = j
    return fc + 128


def _4quantification(d, a):
    """
    对差分值进行量化
    :param d: 差分值
    :param a: 量化因子
    :return: 量化后的值
    """
    fc = 7.5
    if d > 7 * a:
        fc = 7.5
    elif d < -7 * a:
        fc = -7.5
    else:
        for j in range(-8, 8):
            if (j - 1) * a < d <= j * a:
                fc = (2 * j - 1) / 2
                break
    return fc + 7.5


def encapsulation(code):
    """
    将差分数据序列打包
    :param code: 差分数据
    :return: 将差分数据封装之后的数据
    """
    if is8or4 == '8':
        length = len(code)
        w = np.int8(np.ones(length)) * int('11111111', 2)
         for i in range(length):
            w[i] = w[i] & np.int8(code[i])
     else:
        length = len(code)
        w = np.int8(np.ones(length // 2)) * int('11111111', 2)
        for i in range(1, length - 1, 2):
            tmp = w[i // 2] & np.int8(code[i])
            tmp <<= 4
            w[i // 2] = tmp | np.int8(code[i + 1])
     return w


def decode(code, a):
    """
    对差分数据进行解码
    :param code: 差分数据
```

```
    :param a: 量化因子
    :return: 解码结果
    """
    length = len(code)
    x_hat = list(range(length))
    x_hat[0] = np.longlong(code[0])
    for i in range(length - 1):
        if is8or4 == '8':
            # 8bit
            x_hat[i + 1] = np.longlong(x_hat[i]) * u + (code[i + 1] - 128) *
a
        else:
            # 4bit
            x_hat[i + 1] = np.longlong(x_hat[i]) * u + (code[i + 1] - 7.5) *
a
    return x_hat


def save_dpc_file(filename, w, code):
    """
    保存到压缩文件
    :param filename: 文件路径
    :param w: 封装数据
    :param code: 差分数据
    :return: 无
    """
    f = open(filename, 'wb')
    f.write(struct.pack('h', code[0]))
    for i in range(len(w)):
        b = struct.pack('B', w[i])
        f.write(b)
    f.close()


def read_dpc_file(filename):
    """
    读取压缩数据
    :param filename: 文件路径
    :return: 差分数据
    """
    code = []
    with open(filename, 'rb') as f:
        a = struct.unpack('h', f.read(2))
        code.append(a[0])
        while True:
            ff = f.read(1)
            if not ff:
                break
            else:
                a = struct.unpack('B', ff)
                code.append(a[0])
    fcode = [code[0]]
    if is8or4 == '8':
        for i in range(len(code) - 1):
            fcode.append(code[i + 1])
        del fcode[0]
    else:
        for i in range(len(code) - 1):
            fcode.append(code[i + 1] // 16)  # 前4位
            fcode.append(code[i + 1] % 16)  # 后4位
```

```
174.     return fcode
175. 
176. 
177. def save_pcm_file(filename, c):
178.     """
179.     保存到 pcm 文件
180.     :param filename: 文件路径
181.     :param c: 对差分数据进行解码的结果
182.     :return: 无
183.     """
184.     f = wave.open(filename, 'wb')
185.     f.setnchannels(1)
186.     f.setsampwidth(2)
187.     f.setframerate(params[2])
188.     f.writeframes(np.array(c).astype(np.short).tostring())
189.     f.close()
190. 
191. 
192. def snr(data1, data2):
193.     """
194.     计算原数据和解码数据的信噪比
195.     :param data1: 对差分数据进行解码的结果
196.     :param data2: 原数据
197.     :return: 信噪比
198.     """
199.     length = len(data1)
200.     sum1 = np.longlong(0)
201.     sum2 = np.longlong(0)
202.     for i in range(length - 1):
203.         n1 = np.longlong(data1[i]) ** 2 / length
204.         sum1 = sum1 + n1
205.         n2 = np.longlong(data1[i] - data2[i]) ** 2 / length
206.         sum2 = sum2 + n2
207.     return 10 * np.log10(sum1 / sum2)
208. 
209. 
210. if __name__ == '__main__':
211.     for n in range(10):
212.         filepath = './语料/' + str(n + 1) + '.wav'
213.         filepath_dpc = './dpc/' + str(n + 1) + '.dpc'
214.         filepath_pcm = './pcm/' + str(n + 1) + '.pcm'
215. 
216.         # 读入.wav 文件，返回原始数据
217.         wave_data = readfile(filepath)
218. 
219.         # 对原始数据进行 dpcm 压缩
220.         code = dpcm(wave_data, a)
221. 
222.         # 对压缩后的数据进行封装
223.         w = encapsulation(code)
224. 
225.         # 将封装后的数据写入.dpc 文件
226.         save_dpc_file(filepath_dpc, w, code)
227. 
228.         # 从.dpc 文件中读取数据并进行解码
229.         de = decode(read_dpc_file(filepath_dpc), a)
230. 
231.         # 将解码出来的数据写入.pcm 文件
232.         save_pcm_file(filepath_pcm, de)
233. 
```

```
234.         print(a)
235.         print(snr(de, wave_data))
```

## 2.3 解码信号的信噪比

| | |
|---|---|
| 1.wav | 18.79 |
| 2.wav | 14.68 |
| 3.wav | 12.59 |
| 4.wav | 17.87 |
| 5.wav | 18.49 |
| 6.wav | 23.70 |
| 7.wav | 21.94 |
| 8.wav | 17.71 |
| 9.wav | 16.83 |
| 10.wav | 14.14 |

# 三、改进策略

## 3.1 你提出了什么样的改进策略，效果如何

1.修改量化函数

在 4 比特 DPCM 算法中，原先的量化函数为：

$$Q(d,a)=\begin{cases}7\ if\ d>7*a\\-8\ if\ d<-8*a\\j\ if\ (j-1)*a<d\le j*a\end{cases}$$

可以看出输出数据相对于输入偏大，对于均匀分布输出均值大于 0。为了减少量化误差，使得输出均值为 0，可以将输出进行微调如下：

$$Q(d,a)=\begin{cases}7.5\ if\ d>7.5*a\\-7.5\ if\ d<-7.5*a\\\dfrac{2j-1}{2}\ if\ (j-1)*a<d\le j*a\end{cases}$$

编码与解码公式也进行适当修改。

效果如下：

第一种方法，对于 2.wav，计算出的信噪比为 14.66，修改过后信噪比为 14.68。虽然提升很小，但这种方法并不消耗更多的计算量。

2.增加权重

很多的时间序列处理算法中都会在前一个数值的基础上乘一个小于 1 的权重，因此这里使用了增加权重的方法进行实验。

$$d[i]=x[i]-\bar{x}[i-1]*u$$

$$c[i]=Q(d[i],a)$$

$$\bar{x}[i]=\bar{x}[i-1]*u+(c[i]-7.5)*a$$

这里在 $\bar{x}[i-1]$ 的基础上乘以权重 u，相应地，在解码公式中也要乘以权重。

$$\bar{x}[i+1]=\bar{x}[i]*u+(c[i+1]-7.5)*a$$

效果如下：

如果不使用权重，既令 u=1，对于 2.wav，计算出的信噪比为 13.69。如果使用权重进行计算，将 u 设置为 0.91，计算出的信噪比为 14.68。可以看出有较大的提升效果。

# 四、 简述你对量化误差的理解

## 4.1 什么是量化误差？

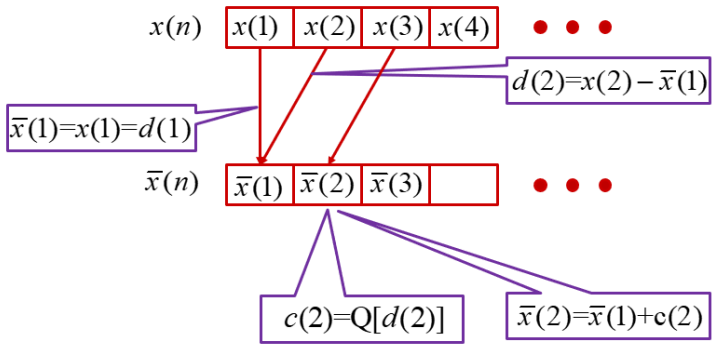
量化误差是指由于对模拟信号进行量化而产生的误差，该误差最大可达到量化等级的一半。量化误差是量化结果和被量化模拟量的差值，量化级数越多，量化的相对误差越小。

一般量化值用 2 禁止表示，如果有 B 个二进制数表示量化值，则一般可以将幅度值划分为 $2^B$ 个等分区间。

从量化过程可以推断出，信号经过量化后，存在一个误差，设 e 为误差，y 是量化后的采样值，即输出，x 为量化前的采样值，即输入。则 e=y-x。

如果信号足够长活量化间隔足够小，则量化误差满足与输入信号无关，且在量化间隔区间内均匀分布。

## 4.2 为什么会编码器中会有一个解码器




在此图中，$\bar{x}$ 为解码结果，即需要一个解码器。在编码时，如果仅仅对相邻的采样值的差值进行编码，则量化误差会被累积叠加到输出信号中，最终导致输出的误差越来越大。为解决这个问题，编码器使用上一次解码后的采样值代替输入来计算差分值，从而减少量化误差叠加带来的不准确。

# 五、 总结

## 5.1 请总结本次实验的收获

本次实验独立完成了 DPCM 编解码算法的编写，对该算法的原理更加熟悉了，了解到各种优化效果的方法，主要是不同的量化方法。
对信噪比这个概念更加了解了。

## 5.2 请给出对本次实验内容的建议

无。