哈爾濱工業大學

实验报告

实 验（二）

题　　目 DPCM 编解码实验

专　　业 计算机科学与技术

学　　号 1160300909

班　　级 1603106

学　　生 张 志 路

指 导 教 师 郑 铁 然

实 验 地 点 G709

实 验 日 期 2018 年 10 月 29 日

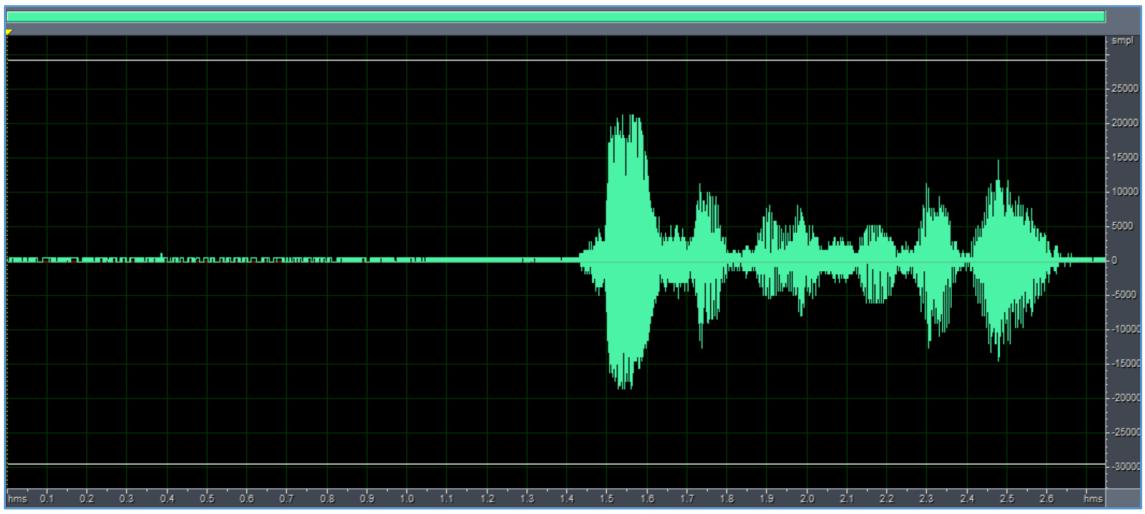
计算机科学与技术学院

目　录

一、 DPCM 编解码算法

1.1 你所采用的量化因子

采用 ppt 所给定的最基本算法进行编解码，当量化因子设置为 593（精确到整数）时，1.wav 文件编解码后的信噪比最高，为 14.426。此时解码后的音频中语音很清晰，但也含有明显噪声。解码后的波形图如下。



报告将在 2.1 部分具体介绍各个改进方法后的量化因子和信噪比。

1.2 拷贝你的算法，加上适当的注释说明

(1) 实验环境

编程语言：python 3.6.4

编程工具：PyCharm 5.0.3、Anaconda3-5.1.0

操作系统：Windows 10 64-bit

(2) 最基本的 DPCM 编码算法的核心部分如下。

```
'''''dpcm 编码'''
def dpcm(waveData, a):
    length = len(waveData)
    code = list(range(length))
    decode = list(range(length))
    code[0] = waveData[0]
    decode[0] = waveData[0]
    for i in range(length-1):
        code[i+1] = waveData[i+1] - decode[i]  # 编码
        code[i+1] = quantification(code[i+1], a)  # 量化
        decode[i+1] = decode[i] + (code[i+1]-8)*a  # 解码，用于反馈
    return code

```

```
'''''封装编码数据，打包'''
def package(code):
    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 = tmp << 4
        w[i//2] = tmp | np.int8(code[i+1])
    return w
```

(3) DPCM 解码算法的核心部分如下。

```
'''''读取编码文件，进行解码'''
def readencode(filename):
    code = []
    with open(filename,'rb') as f:
        a = struct.unpack('h',f.read(2))  # 前两个字节为第一个采样点值
        code.append(a[0])
        while True:
            ff = f.read(1)  # 按字节读
            if not ff:  # 文件末尾
                break
            else:
                a = struct.unpack('B',ff)  # 字节流转换，返回一个元组
                code.append(a[0])
    finalcode = []
    finalcode.append(code[0])
    for i in range(len(code)-1):
        finalcode.append(code[i+1]//16)  # 前 4 位
        finalcode.append(code[i+1]%16)  # 后 4 位
    return finalcode
```

二、改进策略

2.1 你提出了什么样的改进策略，效果如何

2.1.1 修改量化函数

原来的量化函数为 $c(n)=\begin{cases}7 & d(n)>7a\\ -8 & d(n)<-8a\\ i & (i-1)a<d(n)\le ia,-8\le i\le 7\end{cases}$

可以看出，经过此量化器输出后的数据都是相对输入偏大的，此时量化误差较大。为了减小量化误差，使得输出与输入数据差距更小，我采用中点上升量化器，公式如下。

$$c(n)=\begin{cases}7.5 & d(n)>7a\\ -7.5 & d(n)<-7a\\ (2i-1)/2 & (i-1)a<d(n)\le ia,-7\le i\le 7\end{cases}$$

此时的编码和解码公式也应改为如下形式。

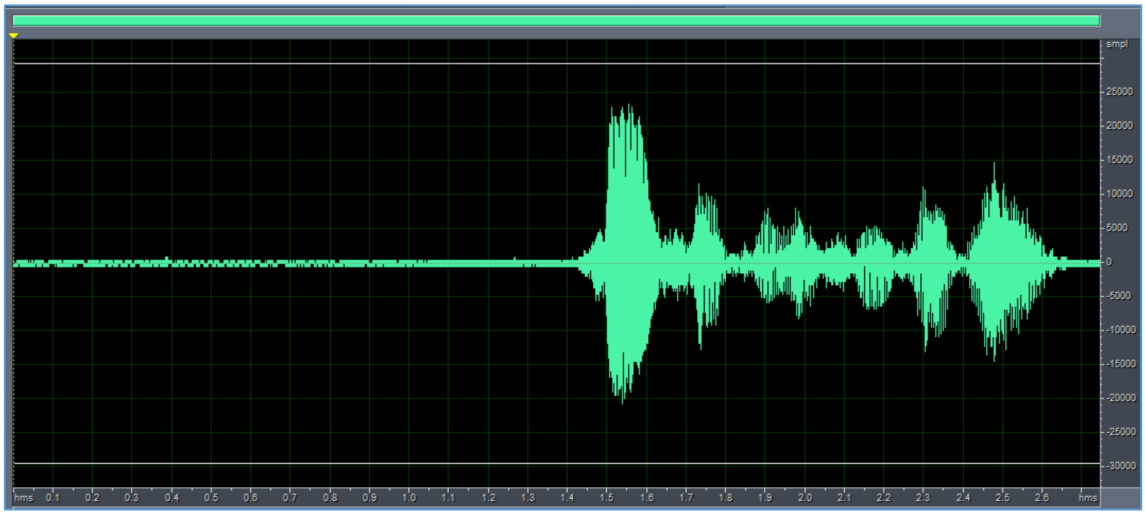
$d(n) = x(n) - \bar{x}(n-1)$

$\bar{x}(n) = \bar{x}(n-1) + (c(n)-7.5)*a)$

经过实验，发现当量化因子取 894（精确到整数）时， 1.wav 编解码后的信噪比达到最大值 17.758。

此时比最基本的方式提高了 3.332dB，音质也有些许提高。

解码后的波形图如下。



2.1.2 取对数

在 2.1.1 的基础上，编码时对原数据取 log 后再做差，修改编码和解码公式分别如下。

$d(n) = \log x(n) - \log \bar{x}(n-1)$

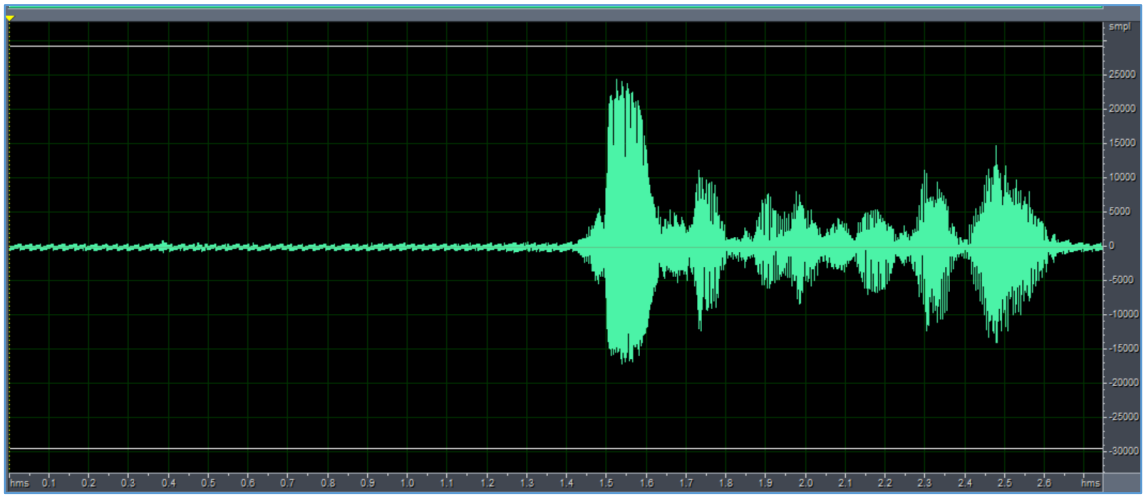
$\bar{x}(n) = \exp(\log \bar{x}(n-1) + (c(n)-7.5)*a)$

此时还需要对数据进行预处理，统一加个值 weight，以防止出现负数，在这里取 weight = - min(data)+100，其中 data 为采样点值列表。同时，在解码时再统一减去 weight 即可。

经过实验，发现当量化因子取 0.0414（精确到小数点后三位）时， 1.wav 编解码后的信噪比达到最大值 18.071。

与 2.1.1 相比，此时信噪比提高地并不明显。解码后的音频也未观察出明显改善。但是前期非语音部分的噪声与 2.1.1 相比似乎更有周期性，显得更加均匀。

解码后的波形图如下。



2.1.3 乘权值

这部分改进的灵感来源于预加重滤波器，即 $H(z) = 1-uz^{-1}$，因此想尝试通过对前一次解码后的采样值乘以一个系数来进行编解码，观察相应效果。

在 2.1.1 的基础上，修改编码和解码公式分别如下。

$d(n) = x(n) - u*\bar{x}(n-1)$

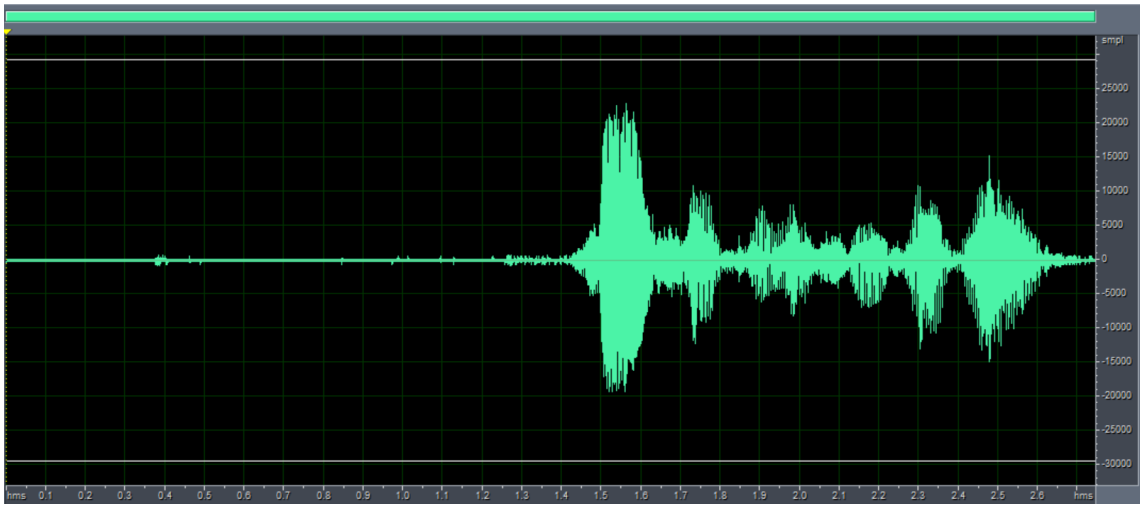
$\bar{x}(n) = u*\bar{x}(n-1) + (c(n)-7.5)*a)$

其中，u 取 0.89-0.99。

经过实验，发现当 u 取 0.91（精确到小数点后两位），量化因子取 869（精确到整数）时，1.wav 编解码后的信噪比达到最大值 18.821。而且此时还有一个现象：信噪比随量化因子变化的幅度相比之前较小。

与 2.1.1 相比，此时信噪比提高 1.063 dB，信噪比提升其实并不明显。但是令人惊喜的是，解码后的音频前期非语音部分的噪声明显减小，语音部分也能感受到噪声的些许减小。

解码后的波形图如下，通过波形图也可以看出前期噪声的明显减小。



2.1.4 前三部分小结

(1) 方法共同点

上面的方法共同点是先对原数据的所有采样点进行 4bit 量化，然后再进行编码，得到的.dpc 压缩文件与原文件的大小比为 1:4，解码后得到的.pcm 文件的大小与原文件相同。

(2) 信噪比与量化字长

查阅资料可知，假设 m^2 表示输入语音信号的方差，n^2 为噪声序列方差，B 为量化字长，X_{max} 为信号峰值。

则量化信噪比 $SNR = 10*\log(m^2/n^2) = 6.02B + 4.77 - 20\log(X_{max}/m)$。

假设语音信号服从拉普拉斯分布，此时信号幅度超过 4m 的概率很小，只有 0.35%，因而可以取 $X_{max} = 4m$，则上式变为：$SNR = 6.02B - 7.2$。此式表明，量化器中每个量化字长对信噪比的贡献大约为 6dB。显然量化字长越大，信噪比越高；当量化字长为 4 时，信噪比大约为 16.88 dB。

(3) 4bit 量化其他改进

上面 2.1.3 中的方法信噪比的最大值已经达到 18.821 dB，如果限制在 4bit 量化，我所能想到的方法中仍可能提高信噪比的就是使用 lpc 或者神经网络预测模型。但在本实验中，限于时间与能力，遗憾的是，我没有成功完成这两项。

除此之外，在使用 4bit 量化时，我还尝试了其他方法。如降采样后 4bit 量化、进一步修改编码和解码公式、运用不同量化方法等，但效果都不理想，因此不再详述。

(4) 更高比特量化

量化字长越大，信噪比越高。当对原数据采用 8bit 量化时，经实验，信噪比最高可以达到 36.47，与理论上的信噪比 $SNR = 6.02B - 7.2 = 40.96$ 差距较大。

当降一半采样后进行 8bit 量化时，信噪比最高可达到 35.62，此时到的.dpc 压缩文件大小仍与 2.1.1-2.1.3 所述方法得到的文件大小相同，即与原文件的大小比为 1:4。但此时解码后得到的.pcm 文件的大小为原文件大小的一半，因为此时有一半采样点被丢掉。

三、 信噪比

3.1 给出所有语音解码还原后的信噪比

以 2.1.3 所述方法为标准，采用 4bit 量化，各个语音解码后的信噪比最大值和所对应的量化因子如下：

文件	量化因子	信噪比
1.wav	869	18.821
2.wav	1531	23.598
3.wav	241	22.699
4.wav	805	18.046
5.wav	972	18.623
6.wav	712	24.621
7.wav	730	22.880
8.wav	844	17.863
9.wav	1103	17.937
10.wav	1018	14.694

对十个量化因子，以信噪比为权值，取加权平均值后得到加权平均量化因子为 874，在该量化因子下各个文件的信噪比如下：

文件	信噪比
1.wav	18.762
2.wav	14.324
3.wav	12.749
4.wav	17.920
5.wav	18.440
6.wav	23.839
7.wav	22.059
8.wav	17.802
9.wav	16.646
10.wav	14.025

可见由于不同语音的特性，信噪比差距较大，十个信噪比的平均值为 17.6566。

四、 简述你对量化误差的理解

4.1 什么是量化误差

一般量化值都用二进制表示，如果 B 个二进制数表示量化值，即量化字长，那么一般将幅度值划分为 2^B 个等分区间。

从量化的过程中可以看出，信号在经过量化后，一定存在一个量化误差。其定义为：e(n) = y(n) – x(n)。

式中，e(n)为量化误差或者噪声；y(n)为量化后的采样值，即量化器的输出；x(n)为未量化的采样值，即量化器的输入。

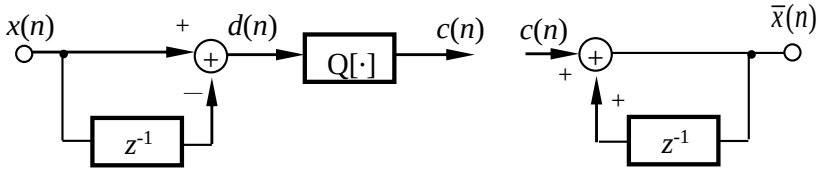
当信号波形足够大或者量化间隔足够小时，可以证明量化噪声符合

① 平稳的白噪声；

② 量化噪声与输入信号不相关；

③ 量化噪声在量化间隔内均匀分布。

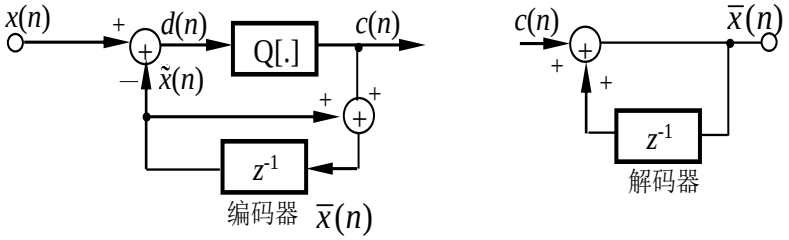
4.2 为什么会编码器中会有一个解码器



编码器　　解码器

在编码时，如果是仅仅对相邻采样值的差值进行编码（如上图），则量化器所产生的量化噪声被累积叠加到了输出信号中，即每次的量化噪声信号都被记录下来，然后叠加到下一次输入中。

为了解决这一问题，编码器应该用前一次解码后的采样值替代前一次的输入采样值，以生成差分信号。如下图所示，编码器通过反馈的方式由差分编码重构生成前一次的采样值。



编码器 $\bar{x}(n)$　　解码器

采用上图结构后，若一个采样点的量化噪声信号为正，则重构后的采样值必将大于输入采样值，在下一时刻，由于使用重构的采样值计算差分，使得差分信号变小而抵消了上一次量化噪声的影响。

总之，DPCM 中编码器内嵌一个解码器的原因就是为了消除量化噪声的累积。

五、 总结

5.1 请总结本次实验的收获

(1) 通过本次实验，我成功设计和完成了一个 DPCM 算法；对 DPCM 的编码和解码原理的认识，也从课堂上的囫囵吞枣到现在的穷原竟委。

(2) 了解了不同的量化方法，并应用于实验中。

(3) 了解了信噪比，编写了计算信噪比的代码，对信噪比的含义有了认识。

5.2 请给出对本次实验内容的建议

无。

参考文献：

[1] 韩纪庆，张磊，郑铁然.语音信号处理[M].北京：清华大学出版社，2013.

附录

以 2.1.3 所述方法为标准，代码如下。

```
import wave
import numpy as np
import struct
params = ()  #文件格式组元
u = 0.91  # 系数权值

'''''读入文件，提取数据'''
def filedata(filename):
    f = wave.open(filename, "rb")
    global params
    params = f.getparams()
    # 返回一个 tuple:
    # (声道数 nchannels,量化位数 sampwidth(byte),采样频率 framerate,采样点数
nframes,...)
    str_data = f.readframes(params[3])
    # readframes 返回的是二进制数据（一大堆 bytes，且不包含文件头)
    f.close()
    wave_data_origin = np.fromstring(str_data, dtype=np.short)
    # 根据声道数和量化单位，将读取的二进制数据转换为一个可以计算的数组
    # 由于我们的声音格式是以两个字节表示一个取样值，因此采用 short 数据类型转换。
    return wave_data_origin

'''''量化'''
def quantification(x, a):
    if x > 7*a:
        final_code = 7.5
    elif x < -7*a:
        final_code = -7.5
    else:
        for j in range(-8,8):
            if x>(j-1)*a and x<=j*a:
                final_code = (2*j-1)/2
                break
    final_code = final_code+7.5
    return final_code

'''''解码'''
def decode(code, a):
    length = len(code)
    decode = list(range(length))
    decode[0] = np.longlong(code[0])
    for i in range(length-1):
        decode[i+1] = np.longlong(decode[i])*u +(code[i+1]-7.5)*a
    return decode

'''''dpcm 编码'''
def dpcm(waveData, a):
    length = len(waveData)
    code = list(range(length))
    decode = list(range(length))
    code[0] = waveData[0]
    decode[0] = waveData[0]
    for i in range(length-1):
```

```
53.         code[i+1] = waveData[i+1] - decode[i]*u  # 编码
54.         code[i+1] = quantification(code[i+1], a)  # 量化
55.         decode[i+1] = decode[i]*u + (code[i+1]-7.5)*a  # 解码，用于反馈
56.     return code
57.
58. '''''封装编码数据，打包'''
59. def package(code):
60.     length = len(code)
61.     w = np.int8(np.ones(length//2))*int('11111111', 2)
62.     for i in range(1, length-1, 2):
63.         tmp = w[i//2] & np.int8(code[i])
64.         tmp = tmp << 4
65.         w[i//2] = tmp | np.int8(code[i+1])
66.     return w
67.
68. '''''保存 dpc 文件'''
69. def savetofile_dpc(filename, w, code):
70.     fw = open(filename, 'wb')
71.     fw.write(struct.pack('h',code[0]))  # 转换为字节流
72.     for i in range(len(w)):
73.         bytes = struct.pack('B',w[i])
74.         fw.write(bytes)
75.     fw.close()
76.
77. '''''读取编码文件，进行解码'''
78. def readencode(filename):
79.     code = []
80.     with open(filename,'rb') as f:
81.         a = struct.unpack('h',f.read(2))  # 前两个字节为第一个采样点值
82.         code.append(a[0])
83.         while True:
84.             ff = f.read(1)  # 按字节读
85.             if not ff:  # 文件末尾
86.                 break
87.             else:
88.                 a = struct.unpack('B',ff)  # 字节流转换，返回一个元组
89.                 code.append(a[0])
90.     finalcode = []
91.     finalcode.append(code[0])
92.     for i in range(len(code)-1):
93.         finalcode.append(code[i+1]//16)  # 前 4 位
94.         finalcode.append(code[i+1]%16)  # 后 4 位
95.     return finalcode
96.
97.
98. '''''写去除静音后的 pcm 语音文件'''
99. def savetofile_pcm(filename,code):
100.     fw = wave.open(filename,'wb')
101.     fw.setnchannels(1) #配置声道数
102.     fw.setsampwidth(2) #配置量化位数
103.     fw.setframerate(params[2]) #配置取样频率
104.     fw.writeframes(np.array(code).astype(np.short).tostring()) #转换为二进制数
    据写入文件
105.     fw.close()
106.
107. '''''计算信噪比'''
108. def snr(data1, data2):
109.     length = len(data1)
110.     sum1 = np.longlong(0)
```

```
    sum2 = np.longlong(0)
    for i in range(length-1):
        n1 = np.longlong(data1[i])**2/length #np.longlong 防止计算时溢出
        sum1 = sum1 + n1
        n2 = np.longlong(data1[i]-data2[i])**2/length
        sum2 = sum2 + n2
    return 10*np.log10(sum1/sum2)

'''''主模块'''
for i in range(1,2,1):
    a = 869  # 量化因子
    filename = '1.wav'
    filename_dpc = '1.dpc'
    filename_pcm = '1.pcm'

    waveData = filedata(filename)  # 读取数据
    m = dpcm(waveData, a)  # 编码
    savetofile_dpc(filename_dpc, package(m), m)  # 存 dpc 压缩文件

    d = decode(readencode(filename_dpc), a)  # 读 dpc 文件，解码
    savetofile_pcm(filename_pcm, d)  # 写解码后的 pcm 文件

    print(a)  # 量化因子
    print(snr(d, waveData))  # 信噪比
```