Binary Numbers

A brief introduction

Why binary?

- Computers use binary to represent numbers
- This is the reason behind the float32, int16 and similar things that you've seen last week
- Binary numbers provide a very terse way of storing simple information numerically
- We need to understand how to decode that information to use the actual data!

How do numbers work?

Take a number, 38748. This means:

100000(10^5)	10000(10^4)	1000(10^3)	100(10^2)	10(10^1)	1 (10^0)
0	3	8	7	4	8

38478 = (8*1) + (4*10) + (7*100) + (8*1000) + (3*10000)

That was Base10

- Typically, we use base 10
- We have an alphabet of 10 symbols
- We use powers of 10
- But this is arbitrary, we could use any other base!

Some other bases

Octal (Base 8): symbols 0 to 8

38748 in octal is: `0113534`

Hexadecimal (Base 16): symbols 0...9, a..f

38748 in hex is `0x964e`

Binary (Base 2): symbols 0 and 1

38748 in binary is `0b1001011001001110`

Binary example

38478 in binary is 0b1001011001001110...

32768	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1
1	**0**	**0**	**1**	**0**	**1**	**1**	**0**	**0**	**1**	**0**	**0**	**1**	**1**	**1**	**0**

We need 16 bits to store that number
What's the smallest integer we can store with 16 bits?
What's the largest integer you can store with 16 bits?

Signed and unsigned integers

We can use one bit to indicate whether a number is positive or negative

Use highest value (**M**ost **S**ignificant **B**it) to store sign (0 for +ve, 1 for -ve)

But we drop 1 bit, so can fit less integers!

Floating point numbers

- So far, we have only considered integers
- How do computers store real numbers?
- As floating point numbers (*floats)*:
  - Store the sign
  - The decimal number
  - And an exponent

Masks

- Binary representation and metadata
- What if we want to set a large number of “flags” (yes/no) concepts for every pixel in a large image?
- We can store them as a binary number, using the position to infer the flag

MODIS LAI QA scheme

7	6	5	4	3	2	1	0
Inversion QA	Inversion QA	Inversion QA	Cloud State	Cloud State	Dead detector	Sensor	Quality

- 5 different flags:
  a. Main quality (1 bit): **Good** or **Bad**
  b. Sensor type (1 bit): **TERRA** or **AQUA**
  c. Cloud state (2 bit): **Clear, significant cloud, mixed clouds, not defined**
  d. Inversion quality (3 bit): **Best possible, very usable, back-up method, back-up method 2, missing**

MODIS LAI QA scheme

7	6	5	4	3	2	1	0	Integer
Inversion QA	Inversion QA	Inversion QA	Cloud State	Cloud State	Dead detector	Sensor	Quality	
0	0	0	0	0	0	0	1	1
1	0	0	0	1	0	1	1	139

- 5 different flags:
  a. Main quality (1 bit): **Good** or **Bad**
  b. Sensor type (1 bit): **TERRA** or **AQUA**
  c. Cloud state (2 bit): **Clear, significant cloud, mixed clouds, not defined**
  d. Inversion quality (3 bit): **Best possible, very usable, back-up method, back-up method 2, missing**

QA masking

- What if we only want to consider a particular flag?
- E.g. `quality_bit == 1`

7	6	5	4	3	2	1	0
Inversion QA	Inversion QA	Inversion QA	Cloud State	Cloud State	Dead detector	Sensor	Quality

QA masking (ii)

- Can **`AND (&)`** by a mask that selects only the wanted bit(s)
- Eg **`(qa_flag & 0b00000001)`**
- If we wanted to select bits 5,6,7?
  - **`temp =( qa_flag & 0b11100000 )`**
  - Then shift bits right by 5 positions:
    - **`np.right_shift( 0b11100000, 5)`**

Recap

- Computers store binary representation
- More bits: more precision, but more storage
- Binary for terse metadata, and usage of flags
- More examples in the class notes!