Machine Learning for Engineering and Science Applications

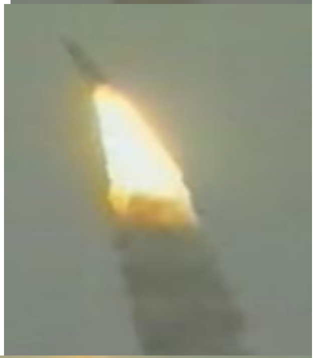
Machine Representation of Numbers
Overflow
Underflow

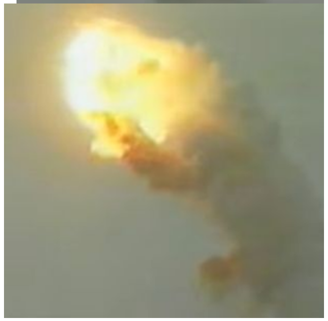
Many of the examples in these slides are from “Applied Numerical Methods” by Steven Chapra

Machine Arithmetic

- The derivations and expressions in the previous slides assume real number arithmetic.
  - That is, all calculations are assumed to take place perfectly
- However, in practice, the machine only stores numbers to a finite precision
- This arithmetic is called “Finite Precision Arithmetic”, “Machine Arithmetic”
  - Or “Floating Point arithmetic” in the special case of floating point numbers
- This can have surprisingly important consequences

The Ariane 5 disaster





- Ariane 5 is a European launch vehicle.
- Its very first test was on June 4^{th}, 1996.
- 37 seconds after launch the rocket turned by 90 degrees incorrectly.
- The boosters were ripped apart and the vehicle self destructed.
- The loss was about $500 million
- The reason was found to be software failure primarily due to forgetting to account for finite precision arithmetic.

https://commons.wikimedia.org/wiki/File:Ariane_5_at_Cite_de_l%27Espace_5.jpg

How a machine stores numbers

- Machines have a finite number of "bits"
  - Think of them as boxes where numbers are stored in binary (0 or 1)
- Computers usually use the binary (Base-2) system
- Integer representation :

$$(10101101)_2 = 2^0 + 2^2 + 2^3 + 2^5 + 2^7 = (173)_{10}$$

1	0	1	0	1	1	0	1

- Signed-integer representation : Use a sign bit (1 for negative)
- Example: With 16-bits, -(173) will be represented as

Magnitude

1	0	0	0	0	0	0	0	1	0	1	0	1	1	0	1

Sign

$$\pm(2^{15} - 1) = \pm 32{,}767$$

This means there is a max and min number that can be represented on the computer

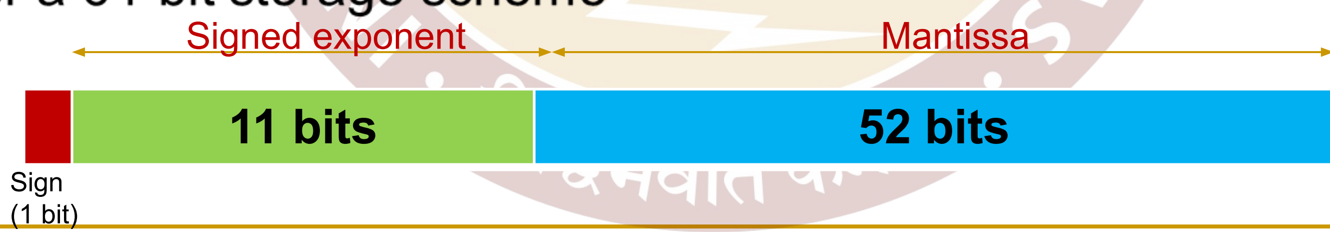
Floating point representation

- Real numbers can also be represented in binary
  - e.g $5.5 = 4 + 0 + 1 + 0.5 = (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0) + (1 \times 2^{-1}) = (101.1)_2$

- The floating-point format is to use the scientific notation $\pm s \times b^e$
  - *S* is the significand, *b* the base we are using, and *e* the exponent
  - Eg : $0.001234 = 1.234 \times 10^{-3}$

- Binary floating point numbers are represented as $\pm\,(1 + f) \times 2^e$
  - Ex : $(5.5)_{10} = (101.1)_2 = 1.011 \times 2^{-2} = (1 + 0.011) \times 2^{-2}$
  - *f* is the mantissa and *e* the exponent
  - Each of *f* and *e* require separate bins to store

- For a 64-bit storage scheme



If you have understood that there is a min and max number on the computer you can skip this slide

Double precision – Range

- IEEE Double precision – 64 bits (8 bytes) are used for storing floating point numbers
  - Is the standard amount of precision used for much of scientific computation
  - MATLAB uses this by default
- Since there are only limited "boxes" for storing the exponent, there is a max/min positive number that can be represented
  - 11 bits for the signed exponent $\Rightarrow$ Range of the exponent is from -1022 to 1023
  - Largest number = $1.111 \ldots 111 \times 2^{+1023} \approx 1.8 \times 10^{308}$
  - Smallest number = $1.000 \ldots 000 \times 2^{-1022} \approx 2.2 \times 10^{-308}$

```
>> format long
>> realmax

ans =

   1.797693134862316e+308

>> realmin

ans =

   2.225073858507201e-308
```

Double precision – Precision

Consider the following example

```
>> sqrt(2)

ans =

  1.414213562373095

>> err = 1.e-14

err =

    1.000000000000000e-14

>> sqrt(2)+err

ans =

  1.414213562373105
```

```
>> err = 1.e-16

err =

    1.000000000000000e-16

>> sqrt(2) +err

ans =

  1.414213562373095
```

```
>> a = 1; b = -1; err = 1.e-16;
>> a+b+err

ans =

    1.000000000000000e-16

>> a+err+b

ans =

   0
```

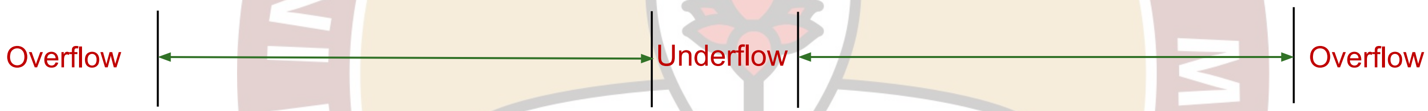
- Happen because even mantissa is limited to 52 bits
- Double Precision (called Machine epsilon) is given by $2^{-52} \approx 2.22 \times 10^{-16}$

```
>> eps

ans =

   2.220446049250313e-16
```

Underflow and overflow

- Underflow happens due to numbers near zero being "rounded off" to zero

```
>> err = 1.e-16

err =

   1.000000000000000e-16

>> sqrt(2) +err

ans =

   1.414213562373095
```



- Underflow might result in divide by zero errors even when the denominator is finite
- Overflow happens when the number being computed overshoots the max possible
- Consider $\frac{\exp(x_1)}{\exp(x_1)+\exp(x_2)}$

```
>> x = [5000 5000];
>> soft = exp(x(1))/(exp(x(1)) + exp(x(2)))

soft =

   NaN
```

```
>> z = x - max(x);
>> soft = exp(z(1))/(exp(z(1)) + exp(z(2)))

soft =

   0.500000000000000
```

- Can be reformulated to account for overflow

The Ariane 5 disaster was due to an overflow problem

The Ariane 5 disaster

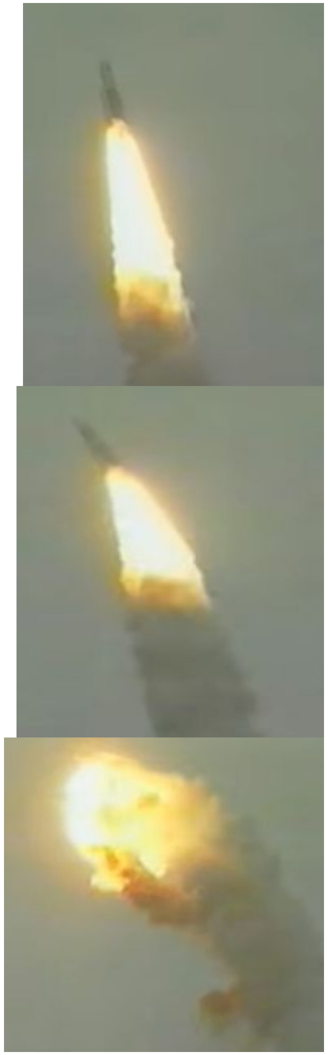
- The internal Inquiry board reported that this was due to software error in the Inertial Reference System (SRI -- Système de Référence Inertielle)

- The SRI used a floating point variable BH to determine the orientation of the rocket.

- This variable was stored as a 64-bit floating point
  - It had to be converted to a 16-bit signed integer

- This caused no problems for the previous version (Ariane 5) as the values were below the max
- After 37s of flight in Ariane 5, these values were exceeded
  - Accelerations were higher in Ariane 5 than Ariane 4

- In the words of the report –
  - The internal SRI software exception was caused during execution of a data conversion from 64-bit floating point to 16-bit signed integer value. The floating point number which was converted had a value greater than what could be represented by a 16-bit signed integer. This resulted in an Operand Error.



http://www-users.math.umn.edu/~arnold/disasters/ariane5rep.html

Condition number

- Limited precision has many unexpected results

```
s=0;
for i = 1:10000
    s = s+0.0001;
end
disp(s)
```

```
>> PrecisionTest
 0.999999999999906
```

Precision effects propagate

- This can be lead to problems when systems of linear equations are solved

```
A =

 1.000000000000000   2.000000000000000
 2.000000000000000   4.000000000100000
```

```
x =

  1
 -1
```

```
>> b = A*x

b =

 -1.000000000000000
 -2.000000000100000
```

```
>> inv(A)*b

ans =

  0.999999999950000
 -0.999999999975000
```

- Some matrices are close to singular
- Small errors can be magnified by them
- This is measured by condition number
- In general $cond(A) = \|A\|\|A^{-1}\|$

```
>> b1

b1 =

 -1.010000000000000
 -2.010000000000000
```

```
>> inv(A)*b1

ans =

  1.0e+08 *

 -1.999999844569314
  0.999999917234647
```

```
>> cond(A)

ans =

   2.499969680591895e+11
```

- For symmetric matrices, condition number is the ratio of the largest to smallest eigenvalue
- $cond(A) = \max_{i,j} \frac{|\lambda_i|}{|\lambda_j|}$. Higher condition number means poorly conditioned