Designs of Algorithms and Programming for Massive Data

DS8001

Andriy Miranskyy

Oct. 3, 2016

Events

- Assignment 1 is posted – due Oct. 17
- Midterm – Oct. 31
  - Time – 6:30-8:30?
  - Note that we will not have office that day.
    - We can discuss a timeslot that would work for everyone closer to the midterm date.
- Final – Dec. 12
  - Tentative date
  - Has to be confirmed by registrar

Outlook

- How does previous lecture helps me with algorithms design?
- Roundoff error and Floating point numbers
- Algebraic Algorithms

How does previous lecture helps me reach course objectives?

- I had the same problem with some of my grad courses.
- A-la: what am I doing here? How is it relevant? At the end, it typically comes together but it may take a while before you get there…

Objectives

- Objective 1: Analyze algorithms and suggest potential ways to parallelize them;
  - Complexity analysis
  - Overall complexity

- Objective 2: Identify existing or design new algorithms required to process massive data.

Greedy heuristics and Subset selection. Example

- We as data scientists are tasked with sampling the data from the large dataset
  - By selecting a subset of variables
- We now know subset selection is a difficult problem (often NP). Chances are – exact solution is not practical.
- We do lit. search and here is what we find:
  - http://link.springer.com/article/10.1007/s10115-014-0801-8

Greedy column subset selection for large-scale data sets

In today's information systems, the availability of massive amounts of data necessitates the development of fast and accurate algorithms to summarize these data and represent them in a succinct format. One crucial problem in big data analytics is the selection of representative instances from large and massively distributed data, which is formally known as the Column Subset Selection problem. The solution to this problem enables data analysts to understand the insights of the data and explore its hidden structure. The selected instances can also be used for data preprocessing tasks such as learning a low-dimensional embedding of the data points or computing a low-rank approximation of the corresponding matrix. This paper presents a fast and accurate greedy algorithm for large-scale column subset selection. The algorithm minimizes an objective function, which measures the reconstruction error of the data matrix based on the subset of selected columns.

…

Greedy column subset selection for large-scale data sets

The paper first presents a centralized greedy algorithm for column subset selection, which depends on a novel recursive formula for calculating the reconstruction error of the data matrix. The paper then presents a MapReduce algorithm, which selects a few representative columns from a matrix whose columns are massively distributed across several commodity machines. The algorithm first learns a concise representation of all columns using random projection, and it then solves a generalized column subset selection problem at each machine in which a subset of columns are selected from the sub-matrix on that machine such that the reconstruction error of the concise representation is minimized. The paper demonstrates the effectiveness and efficiency of the proposed algorithm through an empirical evaluation on benchmark data sets.

Linear (Integer) Programming

- We also learnt about Binary Integer Programming
- Here is how it is used for marketing problems:
  - http://www.maths.ed.ac.uk/~prichtar/Optimization_and_Big_Data/slides/Polik.pdf
    - Slides 16, 17
    - MILP == Mixed-integer linear programming (some variables are integers, some are not)

Representation of numbers

Integer

	Int (long)	Unsigned Int (unsigned long)
Bits	32	32
Sign (length)	1	0
Min integer	$-2^{31} = -2147483648$	0
Max Integer	$2^{31} - 1 = 2147483647$	$2^{32} - 1 = 4294967295$

- R uses 32 bit integers
- With lower level languages you can select the length of your integers
  - byte (8 bits)
  - short (16 bits)
  - long long (64 bits)
  - etc.

Integer: near the limits

- In low-level languages you had to be careful not to overflow
- C code:

```
int x = int(pow(2, 31) - 1);
printf("x = %d, x + 1 = %d\n", x, x + 1);
```

- Output

```
x = 2147483647, x + 1 = -2147483648
```

- Why is it happening?
  - The value “wraps around”

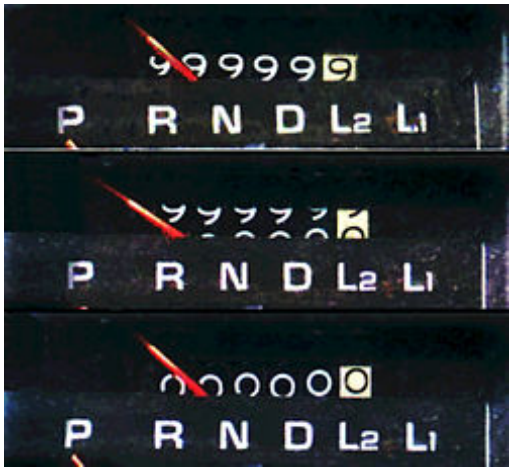


Image taken from: https://en.wikipedia.org/wiki/Integer_overflow#/media/File:Odometer_rollover.jpg

Integer: near the limits

- “On 30 April 2015, the Federal Aviation Authority announced it will order Boeing 787 operators to reset its electrical system periodically, to avoid an integer overflow which could lead to loss of electrical power and ram air turbine deployment, and Boeing is going to deploy a software update in the fourth quarter.[14] The European Aviation Safety Agency followed on 4 May 2015.[15]
- The error happens after 2^{31} centiseconds (248.55134814815 days), indicating a 32-bit signed integer.”

https://en.wikipedia.org/wiki/Integer_overflow

Integer

- R does checking for you (performance overhead)

```
> x <- as.integer(2^31-1); x
[1] 2147483647


> as.integer(x + 1)
[1] NA
Warning message:NAs introduced by coercion to integer
range


> x + 1 #implicit conversion to double
[1] 2147483648
```

Why do I need this?

- Vanilla R has difficulties dealing with large volumes of data
  - If the data cannot be read into memory – you have a problem
- You typically have to resort to
  - packages (e.g. bigmemory) for storing and/or
    - C++
  - external tools (e.g. Spark or H2O) for preprocessing
    - Java / Scala
- Understanding of their implementations becomes important

Floating point number

- Approximation of a real number
- E.g., can’t represent 1/3, pi, sqrt(2), exp(1) in decimal form:
  - $1/3 \approx 0.33333\ldots$

Roundoff error

- Approximation of a real number
  - $1/3 \approx 0.333\ldots$
  - Assuming that I represent the numbers with 3 digits to the right of precision point, the error is
  - $1/3 - 0.333 = 0.0003333\ldots$
- Let's formalize a bit

Fixed point representation

- Assume that we built a number recording system with 3 digits to the left of the precision point and 2 digits to right of the precision point:

  _ _ _ ·_ _

- What is the smallest and the largest numbers that we can record?
  - min – 000.00
  - max – 999.99

Fixed point representation

- I now want to represent the number 142.589 in our fixed system
  - round(142.589, 2) → 142.59
- Absolute Error = |142.589 - 142.59| = 0.001

Fixed point representation

X	round(X, 2)	|X - round(X, 2)|
0.000	0.00	0.000
0.001	0.00	0.001
0.002	0.00	0.002
0.003	0.00	0.003
0.004	0.00	0.004
0.005	**0.01**	**0.005**
0.006	0.01	0.004
0.007	0.01	0.003
0.008	0.01	0.002
0.009	0.01	0.001

In fact the largest absolute error will always be smaller than 0.005 for any number between 0 and 999.99

Fixed point representation

- What about relative error?
- For 142.589:
  - |142.589 - 142.59| / 142.589 = 0.000007
- For 0.589:
  - |0.589 - 0.59| / 0.589 = 0.002
- The smaller the number – the higher the error
- Can we do something about variation of relative error?
  - Floating point representation to the rescue!

Floating point representation

- Scientific notation
  - Move the decimal (radix) point to the right of the first non-zero number:
  - $142.589 \rightarrow 1.42589 \times 10^{2}$
  - $0.589 \rightarrow 5.89 \times 10^{-1}$

- In general,
  - sign x mantissa x 10^{exponent}
    - mantissa is also known as significand
    - exponent is also known as ficand

Floating point representation

- In the fixed system we had five cells:

  _ _ _ . _ _
- Let's reuse the cells but keep the first cell for exponent and the remaining cells for mantissa:

  _ _ _ _ _
- What is the smallest and the largest numbers that can be represented now?
  - Min – 0000 10^0
  - Max – 9999 10^9

Floating point representation

- The range of the numbers:
  - Fixed: 0 – 999
  - Floating: 0 – 9,999,000,000
- The range went up by 7 orders of magnitude
  - $\approx 10^{10}/10^{3}$
- What about the errors?

Floating point representation

- For 142.589:
  - |142.589 – 1.426E2| → 0.011
  - |142.589 – 1.426E2| / 142.589 → 0.0001
- For 123456.789
  - | 123456.789 - 1.234E5 | → 56.789
  - | 123456.789 - 1.234E5 | / 123456.789 → 0.0005

- Variability of relative error went down – it is within the same order

Floating point representation

- How about a sign?
- Naïvely, we need two cells:
  - For the sign of mantissa
  - For the sign of exponent
- However, we can use biased exponent approach to keep only one cell for the sign
  - Set an offset to 5
  - Then the values of exponent will range between -4 (=1-5) and 4 (=9-5)

Double precision

- The same principles apply when going to binary and double precision
  - 64 bits (64 cells taking values {0,1})
  - Sign bit : 1 bit
  - Exponent: 11 bits
  - Mantissa precision: 52 bits
  - $(-1)^{sign}(1.b_{51}b_{50}\ldots b_0) \times 2^{exponent-1023}$
- Compute in R (expected output is 1):
  - $1 + 9E15 - 9E15 \rightarrow 1$
  - $1 + 1E16 - 1E16 \rightarrow 0$
    - Loosing exactness here

Common floating points formats

- Single-precisions a.k.a. float
  - 32 bits
- Double-precision a.k.a. double
  - 64 bits
  - R uses this by default

Float vs. Double

	Float	Double
Length (bits)	32	64
Sign length(bits)	1	1
Exponent length (bits)	8	11
Exponent	-126 to 127	-1022 to 1023
Mantissa length (bits)	23	52
Min number (approx.)	1.2×10^{-38}	2.2×10^{-308}
Max number (approx.)	3.4×10^{38}	1.8×10^{308}

Similar to integers, you can have unsigned float and double – extra bit is added to mantissa

Let’s revisit roundoff error

Roundoff: practical problems

- The timer on the Patriot missile defense system was tracking time in 0.1 second increments. However the time was tracked in a binary approximation of 0.1:
  - 1/10 = 2^-4 + 2^-5 + 2^-8 + 2^-9 + …
  - 24 bit approximation was
    - $0.00011001100110011001100_2$ = 209715 / 2097152
- After 100 hours or 3,600,000 cycles
  - 100 hours * 60 minutes * 60 seconds * 10 “ticks” per second
  - The error accumulated to
  - (1/10 - 209715 / 2097152) * 3600000 ≈ 0.34 seconds
  - Scud travels at ≈ 1,700 meters per second, traveling more than 1/2 km in this timeframe leading to incorrect aim
  - 28 people were killed

Skeel, R. "Roundoff Error and the Patriot Missile." *SIAM News* **25**, 11, Jul. 1992.
https://www.ima.umn.edu/~arnold/disasters/patriot.html

Roundoff: practical problems

- More examples from finance to rocket science: http://mathworld.wolfram.com/RoundoffError.html

Roundoff: practical problems

- When dealing with massive data, the probability of encountering roundoff issue inevitably goes up
- Depending on your goal and setup, it may not be an issue for you, but it is something that you have to be aware of
  - Get summary stats and/or empirical distributions to understand if you may have a problem

Roundoff: practical problems

- It does not have to be summation
- For example, think about Naïve Bayes formula and multiplication. What happens when *p* values are tiny and *n* is very big?

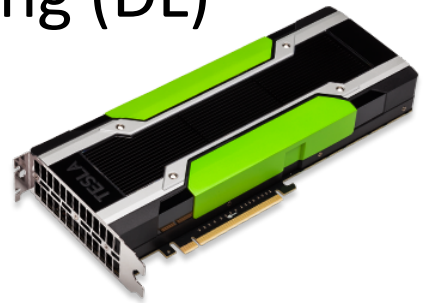
$$p(C_k|x_1,\ldots,x_n) = \frac{1}{Z}p(C_k)\prod_{i=1}^{n}p(x_i|C_k)$$

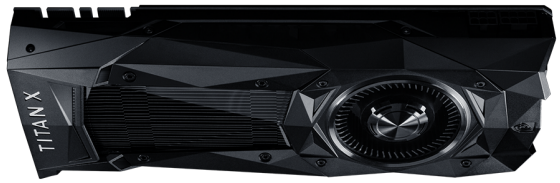
Single vs. Double Precision

- Can store twice as much data in single precision
  - 32 bits vs 64 bits
  - It is important for large volumes of data
- If we are to stick to R then double is our only option…
  - Even though bigmemory package developers may get it in at some point: https://github.com/kaneplusplus/bigmemory/issues/4
- However, there are plenty of other languages out there…

Single vs. Double Precision

- Systems of Partial Differential Equations (PDE) vs. Neural Nets / Deep Learning (DL)





	Nvidia Tesla K80	Nvidia Titan X (Pascal Architecture)
Teraflops single-precision performance	8.73	11
Teraflops double-precision performance	2.91	?
Price (USD)	3750	1200

- *Which card to use for PDE and which one for DL?*

http://www.geforce.com/hardware/10series/titan-x-pascal
http://www.nvidia.com/object/tesla-k80.html

Single vs. Double Precision

- Numerical solutions of PDEs are prone to to roundoff error – go for double precision
- DL are fine with single precision (as shown empirically)
- Thus, it may be cheaper ($) to train DL problems than solve PDEs ☺

Practical Advice

- The majority of modern relational databases and some NoSQL will special decimal data type, where you can specify
  - Maximum number of digits to the left (precision) and right (scale) of precision point
  - This way you can represent, say, financial data and reduce the possibility of round-off error.
  - If you are dealing with currency then most databases will have

http://docs.oracle.com/javadb/10.6.2.1/ref/rrefsqlj15260.html
https://docs.datastax.com/en/cql/3.1/cql/cql_reference/cql_data_types_c.html

If you are interested in number representations on computers

- In-depth discussion is given in D. Knuth, The Art of Computer Programming, Vol. 2, Ch. 4 (Arithmetic)

Truncation error

Truncation error

- Sometimes roundoff error is called truncation error
  - Especially if truncate rather than round the number
- But typically it relates to two other processes:
  - Truncation of infinite series
  - Errors occurring during numerical integration

Ill-conditioned Matrices

Matrices

- Matrices show up in many data analysis problems
  - Linear and non-linear models
  - Linear programming
  - Expected-Maximization
    - Covariance Matrices
  - PageRank
  - etc.

Perturbations

- If there is a small change on RHS, what happens to solution?

$$\begin{cases} x + y = 4, \\ x + 1.001y = 4, \end{cases} \qquad \begin{cases} x + y = 4, \\ x + 1.001y = 4.002, \end{cases}$$

- Solution for the left one is x=4, y =0
- Solution for the right one is x=2, y =2
- A slight perturbation in a single number lead to dramatic change in the answer

What happens if we roundoff the values?

- We may end up getting a very different answer

Condition numbers

- There exists a way to assess how bad the situation is

Condition numbers

- Let's formalize the problem

$$\begin{cases} x_1 + x_2 = 4, \\ x_1 + 1.001x_2 = 4, \end{cases}$$

$$A = \begin{pmatrix} 1 & 1 \\ 1 & 1.001 \end{pmatrix} \qquad x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \qquad b = \begin{pmatrix} 4 \\ 4 \end{pmatrix}$$

- Solve Ax = b

Condition number

- We add a change δb and see how x will be changed (δx)
  - $A(x + \delta x) = b + \delta b$
- We can analyze "robustness" by looking at the condition number κ:
  - $\kappa(A) \geq \frac{\|\delta x\|/\|x\|}{\|\delta b\|/\|b\|}$
  - $\|\cdot\|$ denotes some norm
- The smaller the κ is – the less ill-conditioned the matrix A is
- Ideally, $\kappa = 0$, when $\|\delta x\| = 0$

Example: Empirical

$$A = \begin{pmatrix} 1 & 1 \\ 1 & 1.001 \end{pmatrix}$$

$$b = \begin{pmatrix} 4 \\ 4 \end{pmatrix} \quad \tilde{b} = \begin{pmatrix} 4 \\ 4.002 \end{pmatrix}$$

$$x = \begin{pmatrix} 4 \\ 0 \end{pmatrix} \quad \tilde{x} = \begin{pmatrix} 2 \\ 2 \end{pmatrix}$$

- Using 1-norm
  - $\|b\| = 8$
  - $\|x\| = 4$
  - $\|\delta b\| = \|b - \tilde{b}\| = 0.002$
  - $\|\delta x\| = \|x - \tilde{x}\| = 4$
  - $\kappa(A) \geq \frac{\|\delta x\|/\|x\|}{\|\delta b\|/\|b\|} = \frac{4/4}{0.002/8} = 4000$

Condition number

- An issue with
  - $\kappa(A) \geq \frac{\|\delta x\|/\|x\|}{\|\delta b\|/\|b\|}$
- is that we need to iterate through a lot of $\|\delta b\|$ to get to the upper bound of κ
- It turns out that
  - $\kappa(A) = \frac{\lambda_{max}}{\lambda_{min}}$,
- where λ_{max} and λ_{min} are max and min eigenvalues of A

Example: Eigenvalues

- Eigenvalues of $A = \begin{pmatrix} 1 & 1 \\ 1 & 1.001 \end{pmatrix}$ are 2.000500125 and 0.000499875

- $\kappa(A) = \frac{\lambda_{max}}{\lambda_{min}} = \frac{2.000500125}{0.000499875} = 4002.01$

- The closer the matrix to being singular (non-invertable) the more ill-conditioned the problem is

R: Compute

```
> A <- cbind(1, c(1, 1.001))
> A
     [,1]  [,2]
[1,]    1 1.000
[2,]    1 1.001

> # Direct approach:
> A.eig <- eigen(A)
> max(A.eig$values) / min(A.eig$values)
[1] 4002.001

> # A more sophisticated approach:
> kappa(A)
[1] 4004.001
```

If you are interested to read more about ill-conditioned matrices

- Cleve B. Moler, Numerical Computing with MATLAB, Revised Reprint, 2nd edition, Society for Industrial and Applied Mathematics, 2008. http://www.mathworks.com/moler/chapters.html , Section 2.8
- High-level summary of the eignevalues ratio: http://faculty.nps.edu/rgera/MA3042/2009/ch7.4.pdf
- In-depth analysis: http://math.mit.edu/~edelman/publications/eigenvalues_and_condition.pdf

Practical point

- If R spits a warning saying that the matrix is singular or ill-conditioned, it is a reason to be worried about usefulness of the results

Algebraic (Symbolic) Computations

Computer Algebra: Why?

- We learned that there are a number of potential problems with floating point numbers
  - Can be important for roundoff-related problems, e.g., ill-conditioned matrices
- One of the solutions: computer algebra systems
    - Store irrational and large numbers as-is
    - Keep special functions
    - Solve problems analytically rather than numerically
  - Pros:
    - exact analytic solutions
  - Cons:
    - lack of scalability
    - issues with performance (in comparison with numeric solutions)

Main players

- Maple
  - http://www.maplesoft.com
- Mathematica
  - https://www.wolfram.com/mathematica/
  - Wolfram Alpha uses Mathematica as a backend for math questions https://www.wolframalpha.com
- Matlab Symbolic Toolbox
  - https://www.mathworks.com/products/symbolic/
  - ex-MuPAD
  - Limited functionality in comparison with the first two products

Not to say that we can forget about roundoff errors when dealing with algebraic systems

- Compare output of Mathematica
  - https://www.wolframalpha.com/input/?i=int+x%5E40%2F(x%2B8),+x+%3D+0+..+1
- and Maple
  - http://www.maplesoft.com/applications/view.aspx?SID=4271&view=html
  - The issue is in converting the exact solution expressed in terms of rations to a float number…

Practical advice

- Computer Algebra Systems also help with mundane calculations in algorithmic analysis
- For example, https://www.wolframalpha.com/input/?i=sum(j%2B1,+j+%3D+2..n)
  - Rather than computing by hand – use computer algebra systems

Summary

- How does previous lecture helps me with algorithms design?
- Roundoff error and Floating point numbers
- Algebraic Algorithms
- Labs (D2L → Content → Labs → Lab 4)
  - Memory efficiency
    - How to reduce memory footprint by 99%?
  - Branching optimization
    - Built in functions rule!
  - Roundoff errors
    - Compare summation using floating point algebra and computer algebra

Survey

- Anonymous survey to calibrate the pace of the course
- https://goo.gl/forms/pYPpADd3573eIjoA3