

Algorithms

Arthur J. Redfern

arthur.redfern@utdallas.edu

Feb 11, 2019

Outline

- Motivation
- Sorting
- Application to xNNs
- References

Motivation

Layers And Post Processing

- An algorithm is a set of rules for solving a problem
 - An exceedingly brief look at ~ 1 example for 1 problem is covered here
 - Calling this set of slides “Algorithms” was optimistically 1 letter too many
- Max is a a key component of a common CNN layer and a subset of sorting
 - Max pooling
 - Also as part of spatial pyramid and region of interest pooling variants
- Sorting is a key component of common post processing methods
 - Median and rank order filtering
 - Non maximal suppression

Sorting

Definition

- From Wikipedia: arranging items in a sequence which is ordered based on some criteria

Comparison Sort

- A type of sorting algorithm that applies a comparison operator to elements in a list
- The comparison operator satisfies 2 properties
 - Transitivity: if $a \leq b$ and $b \leq c$ then $a \leq c$
 - Totalness: $\forall a, b$ either $a \leq b$ or $b \leq a$

Comparison Sort

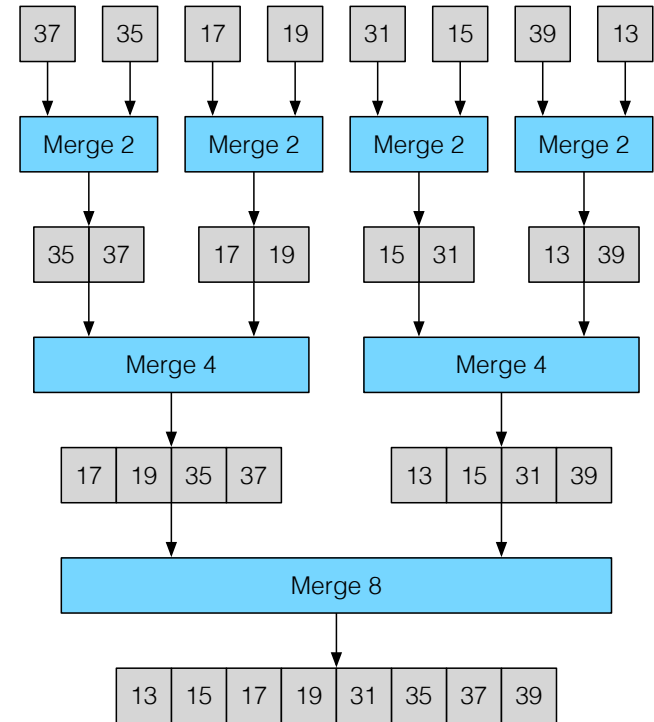
- Optimal comparison sorts (in the sense of minimizing the number of comparisons) require $O(N \log_2(N))$ comparisons where N is the length of the sequence
- A standard (quasi) proof
 - There are $N!$ possible arrangements of a sequence of length N
 - C comparisons can distinguish between 2^C different arrangements
 - To distinguish between all possible arrangements requires $2^C \geq N!$
 - $C \geq \log_2(N!) \approx N \log_2(N) - N \log_2(e) + O(\log_2(N))$, via Stirling's approximation
 $= O(N \log_2(N))$

Comparison Sort

- An information theory based (quasi) proof of the same bound
 - There are $N!$ possible arrangements of a sequence of length N
 - View the arrangements as a random variable $X(s)$
 - The probability of each arrangement is $1/N!$
 - Uniform probability mass function with support of size $N!$
 - The entropy (information) of a realization of this random variable
 - $H(X(s)) = -\sum (1/N!) \log_2(1/N!) = \log_2(N!)$
 - Each comparison in a comparison sort gives at most 1 bit of information
 - To reduce the entropy to 0 with C comparisons need $\log_2(N!) - C \leq 0$
 - $C \geq \log_2(N!) \approx O(N \log_2(N))$

Sequential Merge Sort

- The previous slides discussed comparison sorts and bounds on the minimum number of comparisons in theory
- The sequential merge sort is an example sorting algorithm that achieves the bound in practice
 - Exploits that 2 sorted lists of size $N/2$ can be merged into a sorted list of size N using $N - 1$ comparisons
 - Recursively divides a list into $1/2$ size lists then applies the exploit
 - It's a divide and conquer style algorithm a la the FFT



Sequential Merge Sort

- The number of comparisons in sequential merge sort
 - Depth
 - Total depth $D = \log_2(N)$
 - Depth index $d = 0, \dots, D - 1$
 - Merges
 - Number inputs to a merge at depth d $M_d = 2^{d+1}$
 - Comparisons per merge at depth d $M_d - 1$
 - Comparisons
 - At depth d $(N/M_d) (M_d - 1) = N (M_d - 1) / M_d \approx N$
 - Total $O(N D) = O(N \log_2(N))$

Going Faster: Exploit Information

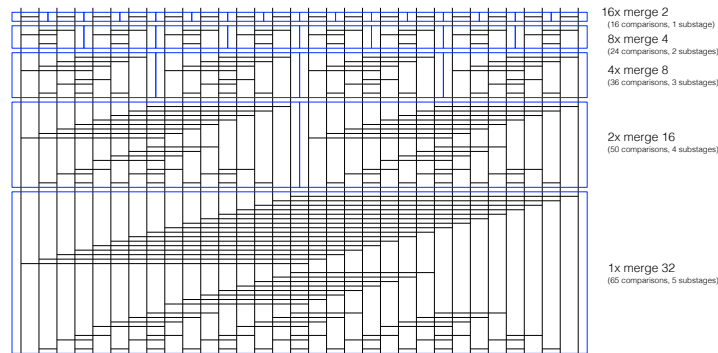
- It's possible to sort a list faster than a comparison sort using **less than** $O(N \log_2(N))$ comparisons if there's known information about the list that's exploitable
 - Remember that a uniform distribution is an entropy maximizing distribution
 - Thinking about sorting from an information theoretic approach, if the probability of the $N!$ possible arrangements is not uniform then $H(X(s)) < \log_2(N!)$ and fewer operations are needed to reduce the information to below 0
 - Maybe apply the known information exploit recursively
 - Maybe clean up approximate arrangement at the end with a comparison sort
- Example
 - Consider sorting 1M last names of random Dallas residents
 - Approximate statistics are known ahead of time as to where a given last name will end up in the final list

Going Faster: Parallel Comparisons

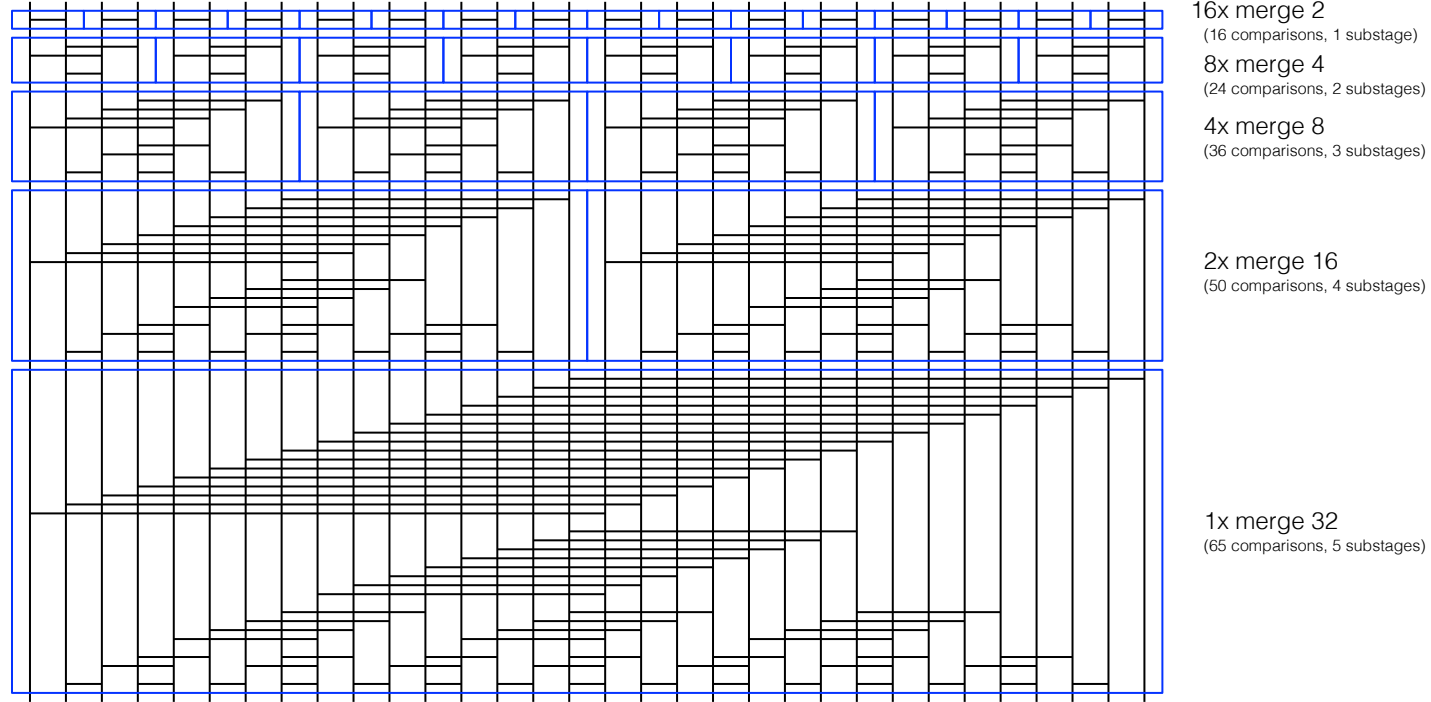
- It's possible to sort a list faster than a comparison sort using **more than** $O(N \log_2(N))$ comparisons (whaaaaaat?)
- A few comments on the direct parallelization of sequential merge sort
 - Separate merges at a given depth are disjoint and can run in parallel (good)
 - Within a merge operation comparisons are done sequentially (bad)
 - Put another way the elements involved in a comparison are dependent on the input (still bad)
- Strategy for going faster on hardware that is optimized for parallel computation: instead of optimizing to minimize the number of comparisons, now optimize to minimize the number of sequential dependencies

Parallel Merge Sort Network

- Sorting networks
 - Loosely: a fixed structure of comparisons (with swaps) that sorts inputs
 - Cookbook rules for creation of some types
 - No input dependence
 - Prove correctness via sorting all 0 / 1 sequences
- Odd even merging network
 - A merge sort style sorting network
 - $O(N \log_2(N) \log_2(N))$ comparisons
 - More than a sequential merge sort
 - $\log_2(N) (\log_2(N) + 1) / 2$ sequential steps
 - Less than a sequential merge sort, this is the key
 - If time: story of Gauss and a misbehaving class



Parallel Merge Sort Network



Application To xNNs

Pooling Layers

- It's common in the encoder portion of CNNs to gradually reduce spatial resolution to increase the receptive field size and reduce the data volume (complexity)
 - CNN style 2D convolution with down sampling (striding) is 1 common way of doing this
 - Pooling is another common way of doing this
- Pooling is a spatial operation that works on individual feature maps (not across feature maps) and maps inputs to down sampled outputs
 - Some variants use max operations and finding the max is a subset of sort hence it's inclusion here
- Common pooling layers (pooling size / stride)
 - Max pooling $3 \times 3 / 2$ and $2 \times 2 / 2$
 - Average pooling $3 \times 3 / 2$ and $2 \times 2 / 2$ // doesn't need sorting
 - Global average pooling $L_r \times L_c / L_r \times L_c$ // doesn't need sorting
 - Spatial pyramid pooling $R_r \times R_c / D_r \times D_c$ to produce a fixed number of $(R_r/D_r)(R_c/D_c)$ elements

Pooling Layers

31	21	33	34	5	2	15
10	29	32	6	27	16	13
7	4	28	20	24	30	26
25	18	14	35	22	1	3
17	23	12	8	19	9	11

Max pool ↓ 3x3 / 2

33	34	30
28	35	30

32	36	68	56	72	48
8	40	64	84	80	12
28	96	92	76	16	4
52	88	44	20	60	24

Avg pool ↓ 2x2 / 2

29	68	53
66	58	26

Pooling Layers

31	21	33	34	5	2	15
10	29	32	6	27	16	13
7	4	28	20	24	30	26
25	18	14	35	22	1	3
17	23	12	8	19	9	11

Max pool ↓ 3x3 / 2

33	34	30
28	35	30

32	36	68	56	72	48
8	40	64	84	80	12
28	96	92	76	16	4
52	88	44	20	60	24

Avg pool ↓ 2x2 / 2

29	68	53
66	58	26

Pooling Layers

31	21	33	34	5	2	15
10	29	32	6	27	16	13
7	4	28	20	24	30	26
25	18	14	35	22	1	3
17	23	12	8	19	9	11

Max pool ↓ 3x3 / 2

33	34	30
28	35	30

32	36	68	56	72	48
8	40	64	84	80	12
28	96	92	76	16	4
52	88	44	20	60	24

Avg pool ↓ 2x2 / 2

29	68	53
66	58	26

Pooling Layers

31	21	33	34	5	2	15
10	29	32	6	27	16	13
7	4	28	20	24	30	26
25	18	14	35	22	1	3
17	23	12	8	19	9	11

Max pool ↓ 3x3 / 2

33	34	30
28	35	30

32	36	68	56	72	48
8	40	64	84	80	12
28	96	92	76	16	4
52	88	44	20	60	24

Avg pool ↓ 2x2 / 2

29	68	53
66	58	26

Median And Rank Order Filtering

- 1D and 2D convolution (correlation)
 - Define filter coefficients $h(\tau)$, $\tau = 0, \dots, L - 1$
 - Generate outputs $y(n)$ from inputs $x(n)$ for the 1D correlation case as

$$y(n) = \sum_{\tau} h(\tau) x(n + \tau), n = 0, \dots, N - L$$

- Linear filter with trainable parameters
- 1D and 2D rank order filtering
 - Define filter length L and rank R
 - Generate outputs $y(n)$ from inputs $x(n)$ for the 1D rank order case as

$$y(n) = \text{select}_R(\text{sort}(x(n), \dots, x(n + L - 1))), n = 0, \dots, N - L$$

- Nonlinear filter with 1 parameter R that selects the R th element from the sorted array

Median And Rank Order Filtering

- Mathematical morphology
 - Erosion
 - 2D rank order filtering where the smallest element is selected
 - Most commonly used on binary images but also applicable to gray scale images
 - Dilation
 - 2D rank order filtering where the largest element is selected
 - Most commonly used on binary images but also applicable to gray scale images
 - Opening
 - Erosion followed by dilation
 - Closing
 - Dilation followed by erosion

Non Maximal Suppression

- Vision
 - Multiple object detection networks frequently have a common convolutional tail and body of a network that takes inputs and generates strong features and 2 heads that work together to do multiple object detection
 - Head 1 to produce region proposals
 - Box coordinates vs anchors and a score indicating likelihood of an object of any class
 - Non maximal suppression can be applied here to keep R best region proposals
 - Head 2 to classifies the R best region proposals each to 1 of C classes
 - Could use spatial pyramid pooling to extract
 - Class 0 ends up with R_0 possibilities (box coordinates + class probability)
 - Class 1 ends up with R_1 possibilities (box coordinates + class probability)
 - ...
 - Non maximal suppression can be applied here to each class individually to keep the N_c best estimates for that class
 - A minimum probability threshold for each class is typically also used

Non Maximal Suppression

- Non maximal suppression algorithm
 - Inputs contain scores and box locations
 - Create a score list by sorting the entries based on the score
 - This is why non maximal suppression is being discussed here
 - Repeat the following until no more entries with scores above a threshold are in the score list
 - Start with the entry with the best score on the score list
 - Remove other entries (suppress non maximals) from the score list with significant overlap
 - Add the best entry from the score list to the prediction list
 - Remove the best entry from the score list
- Variants
 - Can play games like averaging a few together, instead of removing others penalize them by reducing their score, ...
 - A difficulty is finding balance between suppressing windows and detecting close together objects (overlap parameter choice)

References

Sorting

- Sorting networks and their applications
 - <https://dl.acm.org/citation.cfm?id=1468121>
- Batcher's odd-even merging network
 - <http://bekbolatov.github.io/sorting/>
- Batcher's odd-even merging network
 - <http://sparkydots.blogspot.com/2015/05/batchers-odd-even-merging-network.html>

Application To xNNs

- Soft-NMS - Improving Object Detection With One Line of Code
 - <https://arxiv.org/abs/1704.04503>
- Learning non-maximum suppression
 - <https://arxiv.org/pdf/1705.02950.pdf>