Algorithms

Arthur J. Redfern
arthur.redfern@utdallas.edu
Feb 11, 2019

Outline

- Motivation
- Sorting
- Application to xNNs
- Complexity theory
- References

Disclaimer

- This set of slides is more accurately titled “A brief refresher of a subset of algorithms for people already somewhat familiar with the topic followed by it’s specific application to xNN related items needed by the rest of the course”
- However, that’s not very catchy so we’ll just stick with “Algorithms”
- In all seriousness, recognize that algorithms is a very broad and deep topic that has and will continue to occupy many lifetimes of work; if interested in learning more, please consult the references to open a window into a much larger world

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
  - So in addition to the earlier disclaimer, calling this set of slides “Algorithms” was still optimistic by 1 letter too many
- Max is a a key component of a common xNN 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
  - Beam search

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
      $= O(N \log_2(N))$

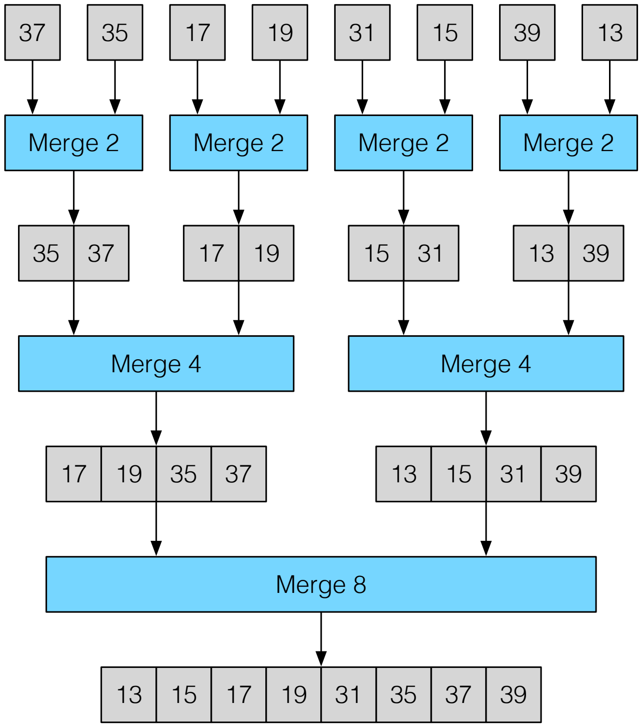
Comparison Sort

- 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
    - $H(X(s)) = -\Sigma\ (1/N!)\ \log_2(1/N!) = \log_2(N!)$
  - 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 $d = 0, ..., D - 1$
  - 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 Known 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

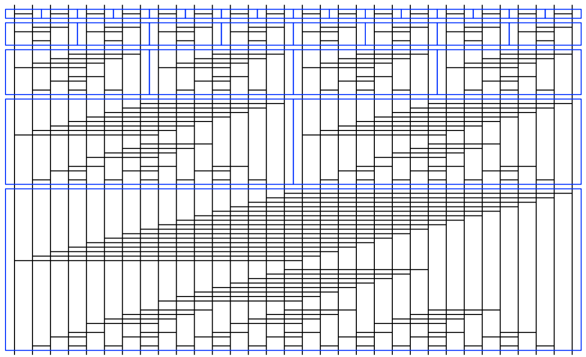
Going Faster: Parallel Comparisons

- It's possible to sort a list faster than a comparison sort using **more than** $O(N \log_2(N))$ comparisons (whaaaaat?)

- 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)

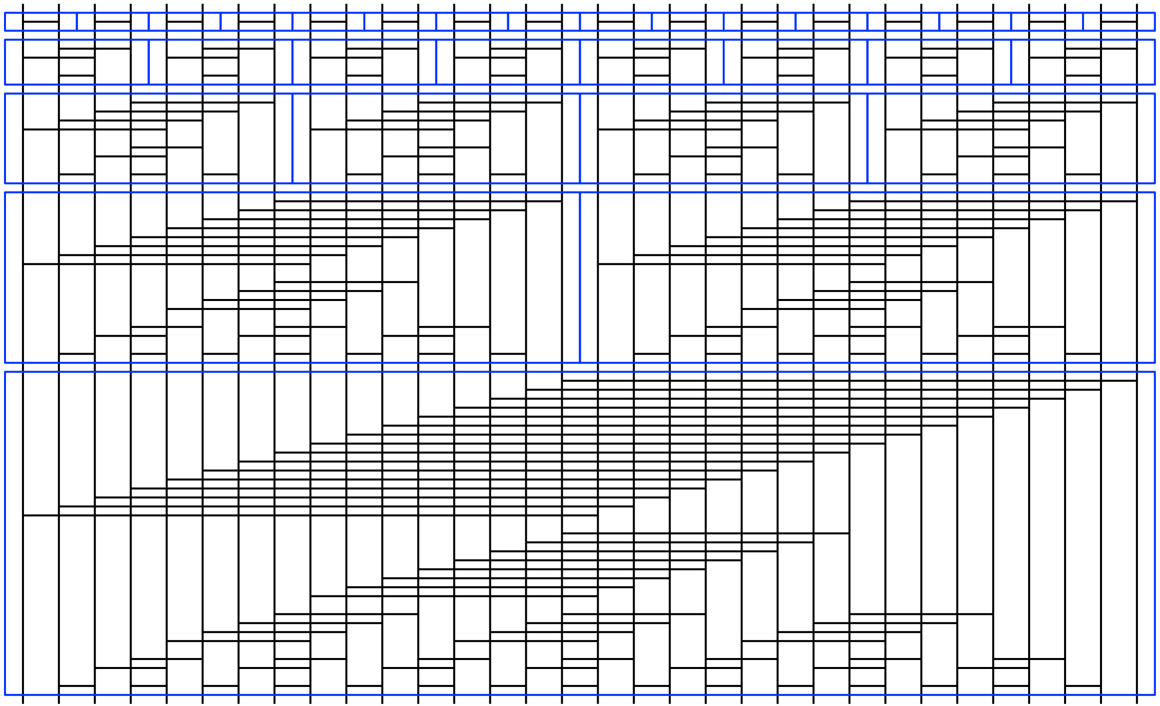
- Strategy for going faster on hardware that is optimized for parallel computation: instead of optimizing to minimize the number of comparisons, now optimize to minimize the number of sequential dependencies

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



16x merge 2
(16 comparisons, 1 substage)

8x merge 4
(24 comparisons, 2 substages)

4x merge 8
(36 comparisons, 3 substages)

2x merge 16
(50 comparisons, 4 substages)

1x merge 32
(65 comparisons, 5 substages)

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 x 3 / 2 and 2 x 2 / 2
  - Average pooling 3 x 3 / 2 and 2 x 2 / 2 // doesn't need sorting
  - Global average pooling L_r x L_c / L_r x L_c // doesn't need sorting
  - Spatial pyramid pooling R_r x R_c / D_r x 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 $\downarrow$ 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 $\downarrow$ 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 $\downarrow$ 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 $\downarrow$ 2x2 / 2

29	68	53
66	58	26

Median And Rank Order Filtering

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

$$y(n) = \Sigma_\tau \, h(\tau) \, x(n + \tau), \; n = 0, \ldots, N - L$$

  - 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

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

  - Nonlinear filter with 1 parameter R that selects the Rth 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

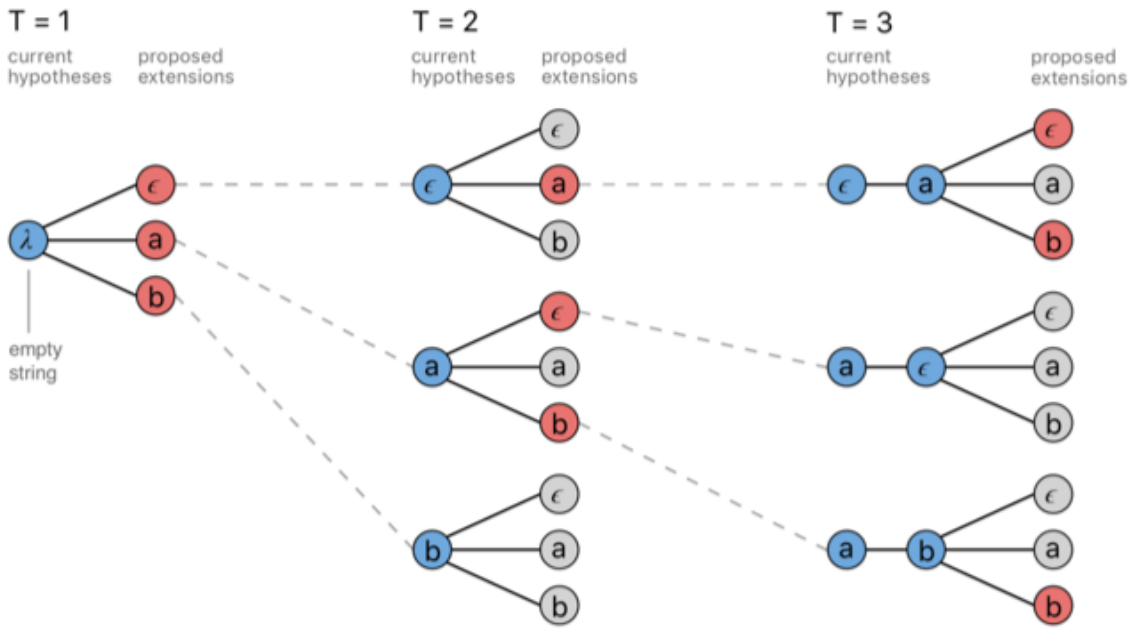
Non Maximal Suppression

A common post processing step in multiple object detection that uses sorting; don’t worry about it here, we’ll cover it in the vision slides

Figure from http://www.telesens.co/2018/03/11/object-detection-and-classification-using-r-cnns/

Beam Search

A common post processing step in speech to text transduction and language translation that uses sorting; don't worry about it here, we'll cover it in the speech and language slides



A standard beam search algorithm with an alphabet of $\{\epsilon, a, b\}$ and a beam size of three.

Figure from https://distill.pub/2017/ctc/

Complexity Theory

Measures

- Many applications of xNNs require lots of memory, data movement and compute
  - Complexity matters to their implementation
  - We're going to count every piece of data and every cycle of compute in the implementation slides
  - So we'll look at complexity from a practical perspective there
- It would be nice to complement the practical measures of complexity that we'll see in the implementation slides with some theoretical measures of complexity here
  - But in doing so we run into the limitations of this being a 1 semester class
  - It would likely take too much time to discuss complexity theory in a meaningful way and the course is already quite dense in terms of topics vs available time
  - So we'll skip talking about complexity theory for now
- For those interested here are a few references on complexity theory as a starting point for going deeper on this topic on your own (likely a complement to information you've learned in previous CS classes)
  - Computational complexity: a modern approach (http://theory.cs.princeton.edu/complexity/book.pdf)
  - Introduction to complexity theory lecture notes (http://www.wisdom.weizmann.ac.il/~oded/PS/CC/all.pdf)
  - The complexity of learning (https://page.mi.fu-berlin.de/rojas/neural/chapter/K10.pdf)

Turing Complete / Computationally Universal

- For the same reasons as not discussing complexity theory (time, literally) we're also not going to discuss Turing completeness other than briefly mentioning it as it's useful to have in the back of your mind when thinking about different xNN architecture options
- Definition (from Wikipedia): "... a system of data-manipulation rules ... is said to be Turing complete or computationally universal if it can be used to simulate any Turing machine"
- A traditional neural network is a universal approximator (shown in the Calculus slides) but is NOT Turing complete
  - This is true of all feed forward neural network structures
- A RNN is Turing complete
  - The key is the feedback / memory
  - Proof: On the computational power of neural nets (https://binds.cs.umass.edu/papers/1992_Siegelmann_COLT.pdf)
  - This is 1 of the motivators behind an interesting line of research on network design that couples a network with an external memory

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