ECE 5755 Fall 2023 Lab 1 #
Nguyen Dang Minh (ndm46)

Problem 1. *Bottleneck of relu and softmax*

Solution. backend bound, followed by front end, is the bottleneck for matmul operations, and the distribution of each component insignificantly changes as the size of the matrices increases.

Problem 2. *Bottleneck for matmul*

Solution. backend bound, followed by front end, is the bottleneck for matmul operations, and the distribution of each component insignificantly changes as the size of the matrices increases.

Problem 3. *Bottleneck of linear*

Solution. backend bound, followed by front end, is the bottleneck for matmul operations, and the distribution of each component insignificantly changes as the size of the matrices increases.

Problem 4. *Bottleneck of conv*

Solution. As the size of the input increases, the amount of computations increased by a bigger order, leading to an increase in back end bound and retiring. Optimization might include improving data locality to avoid long latency data cache misses. I also suspect that retiring takes a longer time because the amount of data allocation to be freed also increases.

Problem 5. *How can you minimize the impact of the code outside of the function you are profiling?*

Solution. Run the function independently with representative input and conditions to measure its performance in isolation. Usage of unit testing frameworks is an option.

Filter out noise (other functions running at the same time) by profiling tool.

Ensure the system is in a consistent state before each run to reduce variability in measurements.

Caching: Use caching to avoid repeated I/O operations.

Problem 6. *Implementation of conv*

Solution. Allocate memories according to the shape as provided by the arguments. Function will return a pointer, to a pointer, to a pointer to a list of float.
Nested for loops: - first iterate through every filter i.e. every filter is calculated one by one - int outputSize = inputSize - kernelSize + 1; output size is calculated as such, now we

iterate through every element of the output, row by row
- for each row, we iterate and calculate the value of each cell (column by column)
- Element wise multiplication from filter and input, then sum up, add bias term
- put the value above through relu activation function and set it to the cell value of the output accordingly.

Problem 7. *Implementation of linear*

Solution. - Allocate memory accordingly. Function will return a pointer to an array of float.
- Calculate each item of output array one by one
- First value is the sum of the first row of weights, times element-wise with input, plus first bias term
- Second value is the sum of the second row of weights, times element-wise with input, plus second bias term
- Carry on this logic.

Problem 8. *Implementation of relu*

Solution. If x is more than 0, return x, else return 0.

Problem 9. *Implementation of matmul*

Solution. Allocate memory accordingly. Result will return pointer to pointer to array of float.
Initialise every element in return value matrix to zero.
Perform matrix multiplication by selecting the right elements to multiply each other and sum up, by first iterating through each row of the left matrix, then iterating through every column of the right matrix, then go down the column (by iterating through every row with a fixed column index) and sum up the element-wise multiplication.

Problem 10. *Implementation of softmax*

Solution. Allocate memory accordingly. Result will return pointer to array of float.
Initialise every element in return value matrix to exponential of input array. Initialise a variable to sum all of these exponential values.
Divide each element in return matrix array by the sum calculated above.

Problem 11. *Implementation of conv test*

Solution. Calculate by hand as well as Python code to cross check, then input result manually to C code, then compare element by element by assert closeness (because float operations are not 100Vary in sizes to check for bottlenecks.
Freeing memory allocations at the end when function concludes.

Problem 12. *Implementation of linear test*

Solution. Calculate by hand as well as Python code to cross check, then input result manually to C code, then compare element by element by assert closeness (because float operations are not 100Vary in sizes to check for bottlenecks.
Freeing memory allocations at the end when function concludes.

Problem 13. *Implementation of matrix ops test*

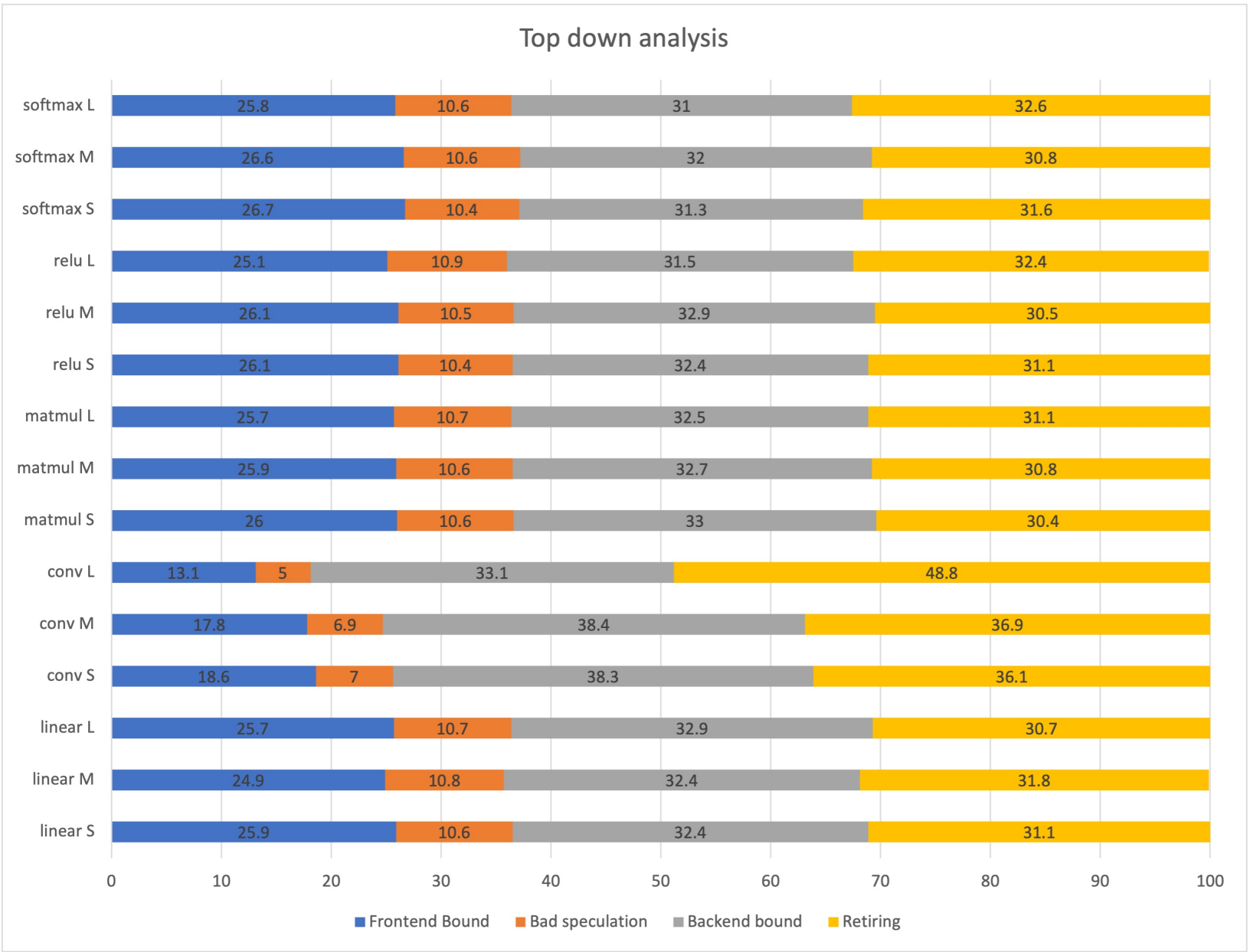
Solution. Calculate by hand as well as Python code to cross check, then input result manually to C code, then compare element by element by assert closeness (because float operations are not 100Vary in sizes to check for bottlenecks. Vary in matrix shapes as well to ensure calculation works as intended even as input matrices are of different shape as long as it satisfies the one condition on shape. (A col = B row)
Freeing memory allocations at the end when function concludes.

Problem 14. *Issues in the lab*

Solution. Wrong index: I needed to ensure that my loops are in the right nested order so that the right elements are accessed. Most of the loops are ¿2 levels deep, this does take up a lot of time.
Similar to the point above, I need to ensure that the shape of the output are correct.
Need to remember to free memory allocations at the end when function concludes.
When I allocated memory to a multidimensional array (matrix), I need to initialise each element to a certain value else it might take a value that was allocated there previously, and it might mess up the returned value because i might perform addition or multiplication with that specific element that has an existing value.
When size of input goes in the 100+ tests take really long.
Took me some time to debug and recall that float operations are approximations so I need to assert value closeness with the correct value as opposed to assert exact equality.

Problem 15. *Suggestions for improvement*

Solution. Need to enforce freeing memories more strictly and thoroughly.


Top down analysis
softmax L
softmax M
softmax S
relu L
relu M
relu S
matmul L
matmul M
matmul S
conv L
conv M
conv S
linear L
linear M
linear S
25.8
10.6
31
32.6
26.6
10.6
32
30.8
26.7
10.4
31.3
31.6
25.1
10.9
31.5
32.4
26.1
10.5
32.9
30.5
26.1
10.4
32.4
31.1
25.7
10.7
32.5
31.1
25.9
10.6
32.7
30.8
26
10.6
33
30.4
13.1
5
33.1
48.8
17.8
6.9
38.4
36.9
18.6
7
38.3
36.1
25.7
10.7
32.9
30.7
24.9
10.8
32.4
31.8
25.9
10.6
32.4
31.1
0
10
20
30
40
50
60
70
80
90
100
Frontend Bound
Bad speculation
Backend bound
Retiring