CS 170 Homework 8

Due **Monday 3/16/2026, at 10:00 pm (grace period until 11:59pm)**

Study Group

List the names and SIDs of the members in your study group. If you have no collaborators, explicitly write "none".

1 FFT in Parallel (Solo Question; 7 points)

We saw in Question 4 on Homework 7 that FFT has an efficient parallel implementation using butterfly circuits, which on length-n inputs has parallel running time $O(\log n)$. Draw the parallel computation dag for the FFT circuit for $n = 8$.

2 Parentheses (10 points)

You are given a string of $2n$ parentheses, e.g. (()(()())), with n open parentheses '(' and n closed parentheses ')'. (You can imagine arithmetic being performed inside the parentheses, but here we are just concerned with the parentheses themselves.) You want to determine if this string represents a valid expression, in which every open parenthesis is later closed. For instance, the string above is valid, while ()))(() is invalid.

Formally, we recursively define a string to be *valid* if every open parenthesis at position i can be matched to a unique close parenthesis at some position $j > i$ such that the substring from $i + 1$ to $j - 1$ is itself valid.

Give a parallel algorithm with $O(n)$ work and $O(\log n)$ time to determine if a given string of parentheses is valid.

Hint: Can you give a condition on the first i parentheses that you can check using parallel prefix, which suffices to check validity?

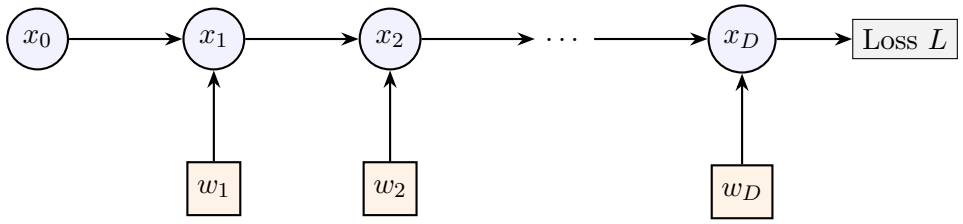
3 Array Compaction (7 points)

Give an efficient parallel algorithm that takes as input an array A of n numbers, and removes all the negative elements. That is, if there are m negative elements, the output has length $n - m$. Analyze T_1 and T_∞.

For example, if $n = 5$ and $A = [2, 5, -3, 6, -7]$, the output should be $[2, 5, 6]$.

4 Stability in Deep Networks (10 points)

Consider a deep network with D layers where all variables are scalars. We assume the network is simply a path, with a single input x_0 and output x_D, and a sequence of weights $w_1, \ldots, w_D$:



In later i, we compute a value x_i as a function of the previous layer's value x_{i-1} along with the weight w_i. We define the loss function as $L = \frac{1}{2}(x_D - y_{target})^2$.

(a) First assume that each layer computes $x_i = w_i \cdot x_{i-1}$. Using the chain rule, compute the derivative $\frac{\partial L}{\partial w_1}$ of the loss with respect to the weight of the first layer.

(b) What happens to your expression from part (a) if $D = 100$ and all weights are initialized to a very small value $w_i \approx 0$? What about $w_i = 0.9$ or $w_i = 1.1$? What would happen if you tried to run gradient descent?

You may assume for simplicity that $|x_0|$ and $|x_D - y_{target}|$ are not too small or large, e.g. between $\frac{1}{5}$ and 5.

(c) To fix this issue, we modify the update rule, so that each layer now computes $x_i = x_{i-1} + F(x_{i-1})$, where $F(x_{i-1}) = w_i \cdot x_{i-1}$. Compute the new gradient $\frac{\partial L}{\partial w_1}$.

(d) Now what happens to your expression from part (c) if $D = 100$ and all weights are initialized to be a very small value $w_i \approx 0$?

(e) Suppose you wanted to initialize the w_i's to be small random values $\pm\epsilon$, so that half of all $w_i = \epsilon$ and the other half are $w_i = -\epsilon$. How large can ϵ be as a function of D to ensure your gradient from part (c) stays stable (neither exploding nor vanishing) as $D \to \infty$?

Note: The rule $x_i = x_{i-1} + F(x_i)$ is called a "residual block", because it only computes the difference, or "residual" between the new value x_i and the previous value x_{i-1}.

5 Gradient Descent in Large Models (5 points)

Modern LLMs have hundreds of billions of parameters, hundreds of layers, and are trained on trillions of tokens. Running gradient descent on a single processor is wildly impractical at this scale. Instead, parallelism is introduced to process the data ("data parallelism"), the model's layers ("pipeline parallelism"), and also the matrix multiplications within a layer ("tensor parallelism").

Look up/ask an AI about how these various forms of parallelism are used together, and write a paragraph reporting your findings.