How to compute row-wise or column-wise aggregations of matrix products in low space

Author: David Xiao

Claps: 82

Date: May 17, 2022

Hereâ€™s a common scenario in data analysis, for example in recommendations: you have two matrices `A` and `B`, and you want to compute some properties of the product matrix `AB`. Because of the way matrix multiplication works, the matrix `AB` may be enormous even if `A` and `B` are not. For example, `A` might have dimensions 10Â¹â�° x 5 and `B` might have dimensions 5 x 10Â¹â�°, then `AB` has dimensions 10Â¹â�° x 10Â¹â�° which is way bigger than either `A` or `B`.

This came up recently in my work collaborating with fellow Klaviyo

Shasha Lin
, where we were trying to compute statistics about the user-item matrix in a collaborative filtering setting. Because Klaviyo works with companies of all sizes, including very large companies with big catalogs and user bases, we sometimes ended up with matrices `AB` that were too large to fit into memory, and so had to look for ways to reduce complexity.

Thereâ€™s not much you can do about this problem for arbitrary properties of `AB`, but if the property you care about behaves nicely, then there may be ways to improve the space/time complexity.

For example, if your goal is to compute the maximum entry in `AB`, then you can basically do this in constant space (assuming for simplicity that storing a real number takes 1 unit of storage): keep a running max (initialized to -âˆž) and compute each entry of `AB` one by one, updating the running max iff the next entry you compute is greater than the previous max.

In this first post, weâ€™ll generalize this space-saving technique for more complicated functions.

Notice however that this technique only saves in space, not in time, which may also be important, especially if this calculation occurs repeatedly in your workflow. In a follow-up post, weâ€™ll also look at probabilistic ways to reduce the time complexity if youâ€™re willing to tolerate a little error.