How to approximate row-wise or column-wise aggregations of matrix products in low space and time

Author: David Xiao

Claps: 7

Date: May 20, 2022

In an earlier post, we explored how to compute row- or column-wise aggregations of matrix products in low space by batching computations. This was motivated by work here at Klaviyo to improve our product recommendations, where we needed to compute properties of the user-item recommendation matrix produced using collaborative filtering.

To recall, we say that a function `F(M)` can be *computed by row-wise aggregation* if there exists a reducer `R` and `initial_value` such that `F` can be computed as follows:

```
import numpy as np
from functools import reduce
from typing import TypeVar, CallableT = TypeVar('T') # generic type for ou
  return reduce(R, M, initial_value)
```

One noticeable shortcoming of the technique described there was that it did not reduce the time complexity of those computations, only the space complexity.

In general there is no perfect way to reduce time complexity because just reading all of the entries of `AB` requires time linear in the size of `AB`, which may already be prohibitive.

One way to get around this is to use probabilistic approximations: for many row-wise aggregation functions, applying the aggregation to a random subsample of the rows of `AB` is a good approximation of the aggregation applied to the entire matrix `AB`. Namely, consider the following `F_approx`:

```
from numpy.random import default_rngrng = default_rng()
def F_approx(A: np.ndarray,
             B: np.ndarray,
             sample_size: int) -> T:
  subsampled_A = rng.choice(A, sample_size, replace=False)
  return F(np.matmul(subsampled_A, B))
```

Weâ€™ll see that in many cases, even for small values of `sample_size`, `F_approx` produces a good approximation to `F`. Obviously, if we include all rows then weâ€™ll get an exact value, while on the other end if we donâ€™t include any rows then we wonâ€™t get any info at all about the value we want to calculate. Our goal is to understand the quantitative tradeoff between these two extremes.

One important caveat: because the notion of what counts as a â€œgood approximationâ€� is highly dependent on the application, there is no one-size-fits-all calculation. Therefore we will focus on two specific examples to highlight some techniques that may be valuable when you analyze your own application: computing the average score of the rows of `AB`, and computing the max of every column of `AB`.