Sparse `einsum`, `bisum`

Julio Candanedo*

August 12, 2023

Abstract

In this work we introduce sparse-tensor contraction method in PyTorch analogous to `einsum` in NumPy.

Statement of Need

Among the many needs in high-performance scientific computing, two major problems arise: we must leverage sparse-data-structures and work with multidimensional-arrays (a parallelizable data-structure). When working with multidimensional-arrays, a clear-and-concise and universal manipulation is the `einsum` function of NumPy, [Harris *et al.*, 2020]. However, there is not much work in the intersection of these needs. I.e. those which manipulate sparse-tensors/arrays as `einsum` does. Therefore this work remedies this need.

Overview of Functionality

As `einsum` stands for Einstein-Summation, `bisum` stands for Binary-Summation. The primary function of this package traces/contractions two tensors at a time (pair sequential-contraction is usually required for efficient contraction in multi-tensor traces) for types: sparse-sparse, sparse-dense, dense-sparse, and dense-dense (the original `einsum` function). This function intakes a string, list-of-tensors, xor tensor; to describe the partial-tracing procedure. Key features include:

1. Efficient Tensor Operations: `bisum` excels in performing a variety of tensor operations, including summation, contraction, and element-wise multiplication, on large dense data structures. While minimizing the memory usage.

2. Sparse Data Focus: the program capitalizes on the idea that many real-world data-sets contain numerous zero values. `bisum` optimizes computations by ignoring these zero values, significantly reducing the computational load and improving execution speed. By eliminating calculations involving zero values, `bisum` reduces memory usage and speeds up computation times, making it a valuable tool for applications involving massive data-sets. This work was originally motivated by [Candanedo, 2023]

3. Streamlined Syntax: `bisum` introduces a user-friendly syntax that simplifies the representation of tensor operations. This enables users to express complex mathematical operations concisely and intuitively, contributing to improved code readability and maintainability.

4. Application Flexibility `bisum` finds applications in a wide range of fields, including scientific research, engineering, machine learning, signal processing, and more. Its efficiency and ease of use make it a versatile choice for various computational tasks. This involves uses in Machine-Learning, Scientific-Simulations (e.g. physics, chemistry, engineering, and etc...), and Signal-Processing.

*`jcandane@asu.edu` / `juliojcandanedo@gmail.com`

5. Optimized for Real Data: While many computations involve zero-padding, `bisum` focuses solely on real data values, eliminating the need to iterate over zero entries. This targeted approach ensures that the program's performance is optimized for sparse/dense, real-valued data-sets. Much real-world data can be made sparse with adequate transformations.

6. Compatibility:`bisum` can be easily integrated into existing code-bases and workflows, complementing other computational libraries and tools. It is integrated with the popular Machine-Learning library PyTorch, [Paszke *et al.*, 2019].

`bisum` fills a crucial niche in the computational landscape by providing a specialized solution for efficient tensor operations on large sparse/dense/mixed data structures. Its focus on sparse data values that significantly boosts performance, making it an indispensable tool for tackling complex calculations in various domains. Whether it's accelerating machine learning tasks or enhancing scientific simulations, `bisum` offers a practical approach to optimizing computations while maintaining code simplicity and readability.

Examples

how to install

`bisum` is on the python-index, and therefore may be easily installed via the following command:

```
pip install bisum
```

import

`bisum` relies on sparse-tensors from PyTorch and therefore we import both libraries as such:

```
import torch
from torch import einsum
from bisum import bisum
```

and on PyTorch we can determine where we would like the tensor to live (on CPU xor GPU)

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

use example

We create natively dense random tensors (and can cast from to be sparse, via the `.to_sparse()` command):

```
A = torch.rand(8**3, device=device).reshape(8,8,8)
B = torch.rand(8**3, device=device).reshape(8,8,8)

torch.allclose( bisum("ijk,kjl", A.to_sparse(), B ), einsum("ijk,kjl", A, B ) )
torch.allclose( bisum("ijk,kjl", A.to_sparse(), B.to_sparse() ).to_dense(),
    einsum("ijk,kjl", A, B ) )
```

brief results when compared to `einsum`

We did a quick comparison of this function (in the sparse-sparse mode) to PyTorch's native `einsum` function. The results of this comparison of relatively sparse-tensors is shown in fig. 1.

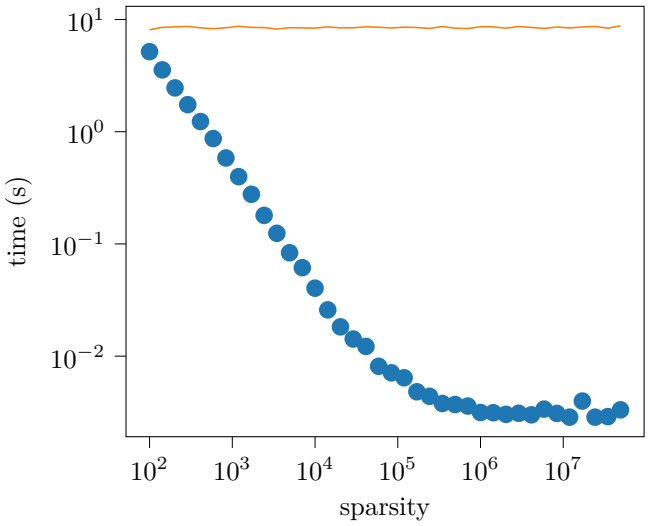

Figure 1: This plot shows a timing comparison between the `torch.einsum` (solid line, averaged over 2 samples) function and the bisum (dots, averaged over 5 samples) function, for the sparse-sparse tensor contraction: $A_{qjwhkrjd}B_{krqljdmn}$ (each tensor of shape $(14 \quad 14 \quad 14 \quad 14 \quad 14 \quad 14 \quad 14 \quad 14)$) on a single CPU.

Development Notes

Currently, `bisum` is in alpha-stage (0.2.0) with code on github: https://github.com/jcandane/bisum, and posted on the Python-Index https://pypi.org/project/bisum/. On here `bisum` has a MIT License. Although, `bisum` is a very useful extension of `einsum` -function, more improvements are desired. Sparse-sparse matrix products on GPUs (tailor-made for dense-dense contractions) are relatively slow, sparse-dense contraction should be much faster. Also functionality on block-sparse or jagged/ragged/PyTree (irregularity shaped) sparse-tensors is desired.

Acknowledgements

This work is supported by Arizona State University Department of Physics and with help from Oliver Beckstein.

References

[Candanedo, 2023] Candanedo, J. J. (2023). *arXiv*. arXiv:2303.10784.

[Harris *et al.*, 2020] Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., and Oliphant, T. E. (2020). *Nature*, 585(7825):357–362.

[Paszke *et al.*, 2019] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., *et al.* (2019). *Advances in Neural Information Processing Systems*, 32.