opt_einsum - A Python package for optimizing contraction order for einsum-like expressions

Daniel G. A. Smith¹ and Johnnie Gray²

¹ The Molecular Science Software Institute, Blacksburg, VA 24060 ² University College London, London, UK

Summary
einsum is a powerful Swiss army knife for arbitrary tensor contractions and general linear algebra found in the popular numpy (Walt, Colbert, and Varoquaux 2011) package. While these expressions can be used to form most mathematical operations found in NumPy, the optimization of these expressions becomes increasingly important as naive implementations increase the overall scaling of these expressions resulting in a dramatic increase in overall execution time. Expressions with many tensors are particularly prevalent in many-body theories such as quantum chemistry, particle physics, and nuclear physics in addition to other fields such as machine learning. At the extreme case, matrix product state theory can have thousands of tensors meaning that the computation cannot proceed in a naive fashion.

The canonical NumPy einsum function considers expressions as a single unit and is not able to factor these expressions into multiple smaller pieces. For example, consider the following index transformation: $M_{pqrs} = C_{pi} C_{qj} I_{ijkl} C_{rk} C_{sl}$ with two different algorithms:

```python
import numpy as np
dim = 10
I = np.random.rand(dim, dim, dim, dim)
C = np.random.rand(dim, dim)

def naive(I, C):
    # N^8 scaling
    return np.einsum('pi,qj,ijkl,rk,sl->pqrs', C, C, I, C, C)

def optimized(I, C):
    # N^5 scaling
    K = np.einsum('pi,ijkl->pjkl', C, I)
    K = np.einsum('qj,pjkl->pqkl', C, K)
    K = np.einsum('rk,pqkl->pqrl', C, K)
    K = np.einsum('sl,pqrl->pqrs', C, K)
    return K
```

By building intermediate arrays the overall scaling of the contraction is reduced and considerable cost savings even for small $N$ ($N=10$) can be seen:

```bash
>> np.allclose(naive(I, C), optimized(I, C))
True
```

1 loops, best of 3: 829 ms per loop

%timeit optimized(I, C)
1000 loops, best of 3: 445 μs per loop

This index transformation is a well known contraction that leads to straightforward intermediates. This contraction can be further complicated by considering that the shape of the C matrices need not be the same, in this case the ordering in which the indices are transformed matters greatly. The opt_einsum package handles this logic automatically and is a drop in replacement for the np.einsum function:

from opt_einsum import contract

dim = 30
I = np.random.rand(dim, dim, dim, dim)
C = np.random.rand(dim, dim)

%timeit optimized(I, C)
10 loops, best of 3: 65.8 ms per loop

%timeit contract('pi,qj,ijkl,rk,sl->pqrs', C, C, I, C, C)
100 loops, best of 3: 16.2 ms per loop

The above automatically will find the optimal contraction order, in this case identical to that of the optimized function above, and computes the products. In this case, it uses np.dot internally to exploit any vendor BLAS functionality that the NumPy build may have.

In addition, backends other than NumPy can be used to either exploit GPU computation via Tensorflow (Abadi et al. 2016) or distributed compute capabilities via Dask (Dask Development Team 2016). The core components of opt_einsum have been contributed back to the numpy.einsum function calls in version 1.12 or later using the optimize keyword (https://docs.scipy.org/doc/numpy-1.14.0/reference/generated/numpy.einsum.html).

The software is on GitHub (https://github.com/dgasmith/opt_einsum/tree/v2.0.0) and can be downloaded via pip or conda-forge. Further discussion of features and uses can be found at the documentation (http://optimized-einsum.readthedocs.io/en/latest/).

Acknowledgements

We acknowledge additional contributions from Fabian-Robert Stöter, Robert T. McGibbon, and Nils Werner to this project.

References
