Linear Operator
Structured Linear Algebra in PyTorch

What is LinearOperator?

LinearOperator is a PyTorch library for structured linear algebra.

High level features:

1. Abstraction: structured matrices are stored as objects that adhere to the torch API but use structure-exploiting algebraic routines under the hood.
2. Composition: Operations on linear operators (e.g., addition, multiplication, Kronecker product, sub-matrix, etc.) produce new LinearOperators that still yield asymptotic savings.
3. Speed/precision tradeoff: Non-structured linear operators can either use exact methods (e.g., Cholesky, QR iterations, etc.) or fast approximate methods (e.g., CG, Lanczos, etc.); all without changing the torch API calls.

How do I use LinearOperator?

LinearOperators look like PyTorch Tensors:

- from linear_operator.operators import DiagLinearOperator, LowRankRootLinearOperator
- C = torch.randn(1000, 200)
- D = torch.randn(1000, 200)
- A = LowRankRootLinearOperator(C) + DiagLinearOperator(D)  # represents C @ CT + diag(d)

Under the hood: The 10 x 10 matrix A is never actually instantiated. It is represented internally by C and d.

They have the same interface as Tensors:
- torch.randn(10, 10)
- torch.add(A, torch.randn(10, 10))  # add b to each entry (element-wise)

Under the hood: LinearOperator objects use __torch_function__ to dispatch all efficient linear algebra operations to the torch namespace.

They exploit encoded structure to make operations efficient:

- b = torch.randn(1000, 1000)
- torch.linalg.solve(A, b)  # compute A^-1 b

Under the hood: Based on known structure, uses the Woodbury formula to compute the solve in O(N) time.

They allow complex getitem operations not supported by other libraries:

- A[0, 1] = torch.randn(1000, 1000)
- A[0:2, 1:3] = torch.randn(2000, 1000)
- Under the hood: LinearOperator postpones instantiating any structured matrices as long as possible. Submatrices (accessed through getitem) are stored as LinearOperators.

Performance Gains

LinearOperator structured representations lead to substantial savings in both time and memory complexity of many operations. Often, lazily evaluating LinearOperators allows to exploit algorithmic structure.

Example 1: Exploiting Kronecker structure

Consider A \(\mathbb{R}^{m \times n}\) and B \(\mathbb{R}^{n \times p}\) matrices. The Kronecker product of A and B is an \(m n \times np\) matrix \(A \otimes B\). LinearOperator constructs \(A \otimes B\) eagerly. However, LinearOperator implements efficient algorithms for many linear algebra operations on Kronecker products, including matrix multiplication, computing spectral properties, etc.

Example 2: Monarch Matrices

Monarch matrices are a recently-proposed class of structured matrices that allows for fast matrix-vector products and thereby, efficient neural network training and inference. LinearOperator enables a fast and easy implementation of Monarch matrices:

- P = PermutationLinearOperator(torch.randperm(n))
- L = BlockDiagLinearOperator(torch.randn(n, n), torch.randn(n, n))
- R = (P @ L @ P.T) @ R

High-performance iterative methods

LinearOperator implements efficient iterative methods to approximately solve linear systems of equations, eigenvalue problems and more. On large and structured matrices, this can lead to large speed ups.

Plug-in speedups

LinearOperator objects can be used as drop-in replacements for regular tensors in existing code, and algebraic structure will be exploited automatically.

Limitations of LinearOperator

- LinearOperator is implemented in pure python. If the represented tensors are small, then the python overhead can end up making LinearOperator slower than working with dense (small) tensors.
- Currently, LinearOperator focuses primarily on symmetric, positive definite matrices and operator coverage for general tensors is limited. We would love to change that with the help of the community.

Acknowledgements

Geoff Pleiss is supported by the Simons Foundation, McKnight Foundation, Grossman Center for the Statistics of Mind, and Gatsby Charitable Trust. Jacob Gardner is supported by NSF award IIS-2145644.

https://github.com/cornellius-gp/linear_operator