

tf.contrib.linalg.LinearOperatorFullMatrix

Contents

Class `LinearOperatorFullMatrix`

Properties

`batch_shape``domain_dimension`Class **LinearOperatorFullMatrix**Inherits From: [LinearOperator](#)Defined in [tensorflow/contrib/linalg/python/ops/linear_operator_full_matrix.py](#).See the guide: [Linear Algebra \(contrib\)](#) > [LinearOperator](#)**LinearOperator** that wraps a [batch] matrix.

This operator wraps a [batch] matrix **A** (which is a **Tensor**) with shape **[B1,...,Bb, M, N]** for some **b >= 0**. The first **b** indices index a batch member. For every batch index **(i1,...,ib)**, **A[i1,...,ib, : :]** is an **M x N** matrix.

```
# Create a 2 x 2 linear operator.
matrix = [[1., 2.], [3., 4.]]
operator = LinearOperatorFullMatrix(matrix)

operator.to_dense()
==> [[1., 2.]
      [3., 4.]]

operator.shape
==> [2, 2]

operator.log_abs_determinant()
==> scalar Tensor

x = ... Shape [2, 4] Tensor
operator.matmul(x)
==> Shape [2, 4] Tensor

# Create a [2, 3] batch of 4 x 4 linear operators.
matrix = tf.random_normal(shape=[2, 3, 4, 4])
operator = LinearOperatorFullMatrix(matrix)
```

Shape compatibility

This operator acts on [batch] matrix with compatible shape. **x** is a batch matrix with compatible shape for **matmul** and **solve** if

```
operator.shape = [B1,...,Bb] + [M, N], with b >= 0
x.shape = [B1,...,Bb] + [N, R], with R >= 0.
```

Performance

`LinearOperatorFullMatrix` has exactly the same performance as would be achieved by using standard `TensorFlow` matrix ops. Intelligent choices are made based on the following initialization hints.

- If `dtype` is real, and `is_self_adjoint` and `is_positive_definite`, a Cholesky factorization is used for the determinant and solve.

In all cases, suppose `operator` is a `LinearOperatorFullMatrix` of shape `[M, N]`, and `x.shape = [N, R]`. Then

- `operator.matmul(x)` is $O(M * N * R)$.
- If `M=N`, `operator.solve(x)` is $O(N^3 * R)$.
- If `M=N`, `operator.determinant()` is $O(N^3)$.

If instead `operator` and `x` have shape `[B1, ..., Bb, M, N]` and `[B1, ..., Bb, N, R]`, every operation increases in complexity by `B1*...*Bb`.

Matrix property hints

This `LinearOperator` is initialized with boolean flags of the form `is_X`, for `X = non_singular, self_adjoint, positive_definite, square`. These have the following meaning:

- If `is_X == True`, callers should expect the operator to have the property `X`. This is a promise that should be fulfilled, but is *not* a runtime assert. For example, finite floating point precision may result in these promises being violated.
- If `is_X == False`, callers should expect the operator to not have `X`.
- If `is_X == None` (the default), callers should have no expectation either way.

Properties

batch_shape

`TensorShape` of batch dimensions of this `LinearOperator`.

If this operator acts like the batch matrix `A` with `A.shape = [B1, ..., Bb, M, N]`, then this returns `TensorShape([B1, ..., Bb])`, equivalent to `A.get_shape()[:-2]`

Returns:

`TensorShape`, statically determined, may be undefined.

domain_dimension

Dimension (in the sense of vector spaces) of the domain of this operator.

If this operator acts like the batch matrix `A` with `A.shape = [B1, ..., Bb, M, N]`, then this returns `N`.

Returns:

`Dimension` object.

dtype

The `DType` of `Tensor` s handled by this `LinearOperator` .

`graph_parents`

List of graph dependencies of this `LinearOperator` .

`is_non_singular`

`is_positive_definite`

`is_self_adjoint`

`is_square`

Return `True/False` depending on if this operator is square.

`name`

Name prepended to all ops created by this `LinearOperator` .

`range_dimension`

Dimension (in the sense of vector spaces) of the range of this operator.

If this operator acts like the batch matrix `A` with `A.shape = [B1, ..., Bb, M, N]` , then this returns `M` .

Returns:

`Dimension` object.

`shape`

`TensorShape` of this `LinearOperator` .

If this operator acts like the batch matrix `A` with `A.shape = [B1, ..., Bb, M, N]` , then this returns `TensorShape([B1, ..., Bb, M, N])` , equivalent to `A.get_shape()` .

Returns:

`TensorShape` , statically determined, may be undefined.

`tensor_rank`

Rank (in the sense of tensors) of matrix corresponding to this operator.

If this operator acts like the batch matrix `A` with `A.shape = [B1, ..., Bb, M, N]` , then this returns `b + 2` .

Args:

- `name` : A name for this `Op`.

Returns:

Python integer, or None if the tensor rank is undefined.

Methods

`__init__`

```
__init__(
    matrix,
    is_non_singular=None,
    is_self_adjoint=None,
    is_positive_definite=None,
    is_square=None,
    name='LinearOperatorFullMatrix'
)
```

Initialize a `LinearOperatorFullMatrix`.

Args:

- `matrix`: Shape `[B1,...,Bb, M, N]` with `b >= 0`, `M, N >= 0`. Allowed dtypes: `float32`, `float64`, `complex64`, `complex128`.
- `is_non_singular`: Expect that this operator is non-singular.
- `is_self_adjoint`: Expect that this operator is equal to its hermitian transpose.
- `is_positive_definite`: Expect that this operator is positive definite, meaning the quadratic form $\mathbf{x}^H \mathbf{A} \mathbf{x}$ has positive real part for all nonzero \mathbf{x} . Note that we do not require the operator to be self-adjoint to be positive-definite. See: https://en.wikipedia.org/wiki/Positive-definite_matrix\#Extension_for_non_symmetric_matrices
- `is_square`: Expect that this operator acts like square [batch] matrices.
- `name`: A name for this `LinearOperator`.

Raises:

- `TypeError`: If `diag.dtype` is not an allowed type.

`add_to_tensor`

```
add_to_tensor(
    x,
    name='add_to_tensor'
)
```

Add matrix represented by this operator to `x`. Equivalent to `A + x`.

Args:

- `x`: `Tensor` with same `dtype` and shape broadcastable to `self.shape`.
- `name`: A name to give this `Op`.

Returns:

A `Tensor` with broadcast shape and same `dtype` as `self`.

assert_non_singular

```
assert_non_singular(name='assert_non_singular')
```

Returns an **Op** that asserts this operator is non singular.

This operator is considered non-singular if

```
ConditionNumber < max{100, range_dimension, domain_dimension} * eps,  
eps := np.finfo(self.dtype.as_numpy_dtype).eps
```

Args:

- **name** : A string name to prepend to created ops.

Returns:

An **Assert Op**, that, when run, will raise an **InvalidArgumentError** if the operator is singular.

assert_positive_definite

```
assert_positive_definite(name='assert_positive_definite')
```

Returns an **Op** that asserts this operator is positive definite.

Here, positive definite means that the quadratic form $\mathbf{x}^H \mathbf{A} \mathbf{x}$ has positive real part for all nonzero \mathbf{x} . Note that we do not require the operator to be self-adjoint to be positive definite.

Args:

- **name** : A name to give this **Op**.

Returns:

An **Assert Op**, that, when run, will raise an **InvalidArgumentError** if the operator is not positive definite.

assert_self_adjoint

```
assert_self_adjoint(name='assert_self_adjoint')
```

Returns an **Op** that asserts this operator is self-adjoint.

Here we check that this operator is *exactly* equal to its hermitian transpose.

Args:

- **name** : A string name to prepend to created ops.

Returns:

An **Assert Op**, that, when run, will raise an **InvalidArgumentError** if the operator is not self-adjoint.

batch_shape_tensor

```
batch_shape_tensor(name='batch_shape_tensor')
```

Shape of batch dimensions of this operator, determined at runtime.

If this operator acts like the batch matrix **A** with **A.shape** = **[B1, ..., Bb, M, N]**, then this returns a **Tensor** holding **[B1, ..., Bb]**.

Args:

- **name**: A name for this `Op`.

Returns:

int32 Tensor

determinant

```
determinant(name='det')
```

Determinant for every batch member.

Args:

- **name**: A name for this `Op`.

Returns:

Tensor with shape **self.batch_shape** and same **dtype** as **self**.

Raises:

- **NotImplementedError**: If **self.is_square** is **False**.

diag_part

```
diag_part(name='diag_part')
```

Efficiently get the [batch] diagonal part of this operator.

If this operator has shape **[B1, ..., Bb, M, N]**, this returns a **Tensor diagonal**, of shape **[B1, ..., Bb, min(M, N)]**, where **diagonal[b1, ..., bb, i] = self.to_dense()[b1, ..., bb, i, i]**.

```
my_operator = LinearOperatorDiag([1., 2.])

# Efficiently get the diagonal
my_operator.diag_part()
==> [1., 2.]

# Equivalent, but inefficient method
tf.matrix_diag_part(my_operator.to_dense())
==> [1., 2.]
```

Args:

- `name` : A name for this `Op` .

Returns:

- `diag_part` : A `Tensor` of same `dtype` as self.

domain_dimension_tensor

```
domain_dimension_tensor(name='domain_dimension_tensor')
```

Dimension (in the sense of vector spaces) of the domain of this operator.

Determined at runtime.

If this operator acts like the batch matrix `A` with `A.shape = [B1, ..., Bb, M, N]` , then this returns `N` .

Args:

- `name` : A name for this `Op` .

Returns:

```
int32 Tensor
```

log_abs_determinant

```
log_abs_determinant(name='log_abs_det')
```

Log absolute value of determinant for every batch member.

Args:

- `name` : A name for this `Op`.

Returns:

`Tensor` with shape `self.batch_shape` and same `dtype` as `self` .

Raises:

- `NotImplementedError` : If `self.is_square` is `False` .

matmul

```
matmul(  
    x,  
    adjoint=False,  
    adjoint_arg=False,  
    name='matmul'  
)
```

Transform [batch] matrix \mathbf{x} with left multiplication: $\mathbf{x} \rightarrow \mathbf{Ax}$.

```
# Make an operator acting like batch matrix A. Assume A.shape = [..., M, N]
operator = LinearOperator(...)
operator.shape = [..., M, N]

X = ... # shape [..., N, R], batch matrix, R > 0.

Y = operator.matmul(X)
Y.shape
==> [..., M, R]

Y[..., :, r] = sum_j A[..., :, j] X[j, r]
```

Args:

- \mathbf{x} : **Tensor** with compatible shape and same **dtype** as **self**. See class docstring for definition of compatibility.
- **adjoint**: Python **bool**. If **True**, left multiply by the adjoint: $\mathbf{A}^H \mathbf{x}$.
- **adjoint_arg**: Python **bool**. If **True**, compute $\mathbf{A} \mathbf{x}^H$ where \mathbf{x}^H is the hermitian transpose (transposition and complex conjugation).
- **name**: A name for this `Op`.

Returns:

A **Tensor** with shape $[\dots, \mathbf{M}, \mathbf{R}]$ and same **dtype** as **self**.

matvec

```
matvec(
    x,
    adjoint=False,
    name='matvec'
)
```

Transform [batch] vector \mathbf{x} with left multiplication: $\mathbf{x} \rightarrow \mathbf{Ax}$.

```
# Make an operator acting like batch matrix A. Assume A.shape = [..., M, N]
operator = LinearOperator(...)

X = ... # shape [..., N], batch vector

Y = operator.matvec(X)
Y.shape
==> [..., M]

Y[..., :] = sum_j A[..., :, j] X[..., j]
```

Args:

- \mathbf{x} : **Tensor** with compatible shape and same **dtype** as **self**. \mathbf{x} is treated as a [batch] vector meaning for every set of leading dimensions, the last dimension defines a vector. See class docstring for definition of compatibility.
- **adjoint**: Python **bool**. If **True**, left multiply by the adjoint: $\mathbf{A}^H \mathbf{x}$.
- **name**: A name for this `Op`.

Returns:

A `Tensor` with shape `[..., M]` and same `dtype` as `self`.

range_dimension_tensor

```
range_dimension_tensor(name='range_dimension_tensor')
```

Dimension (in the sense of vector spaces) of the range of this operator.

Determined at runtime.

If this operator acts like the batch matrix `A` with `A.shape = [B1, ..., Bb, M, N]`, then this returns `M`.

Args:

- `name`: A name for this `Op`.

Returns:

`int32 Tensor`

shape_tensor

```
shape_tensor(name='shape_tensor')
```

Shape of this `LinearOperator`, determined at runtime.

If this operator acts like the batch matrix `A` with `A.shape = [B1, ..., Bb, M, N]`, then this returns a `Tensor` holding `[B1, ..., Bb, M, N]`, equivalent to `tf.shape(A)`.

Args:

- `name`: A name for this `Op`.

Returns:

`int32 Tensor`

solve

```
solve(  
    rhs,  
    adjoint=False,  
    adjoint_arg=False,  
    name='solve'  
)
```

Solve (exact or approx) `R` (batch) systems of equations: `A X = rhs`.

The returned `Tensor` will be close to an exact solution if `A` is well conditioned. Otherwise closeness will vary. See class docstring for details.

Examples:

```
# Make an operator acting like batch matrix A. Assume A.shape = [..., M, N]
operator = LinearOperator(...)
operator.shape = [..., M, N]

# Solve  $R > 0$  linear systems for every member of the batch.
RHS = ... # shape [..., M, R]

X = operator.solve(RHS)
#  $X[... , :, r]$  is the solution to the  $r$ 'th linear system
#  $\sum_j A[... , :, j] X[... , j, r] = RHS[... , :, r]$ 

operator.matmul(X)
==> RHS
```

Args:

- **rhs**: **Tensor** with same **dtype** as this operator and compatible shape. **rhs** is treated like a [batch] matrix meaning for every set of leading dimensions, the last two dimensions defines a matrix. See class docstring for definition of compatibility.
- **adjoint**: Python **bool**. If **True**, solve the system involving the adjoint of this **LinearOperator**: $A^H X = rhs$.
- **adjoint_arg**: Python **bool**. If **True**, solve $A X = rhs^H$ where rhs^H is the hermitian transpose (transposition and complex conjugation).
- **name**: A name scope to use for ops added by this method.

Returns:

Tensor with shape $[..., N, R]$ and same **dtype** as **rhs**.

Raises:

- **NotImplementedError**: If **self.is_non_singular** or **is_square** is False.

solvevec

```
solvevec(
    rhs,
    adjoint=False,
    name='solve'
)
```

Solve single equation with best effort: $A X = rhs$.

The returned **Tensor** will be close to an exact solution if **A** is well conditioned. Otherwise closeness will vary. See class docstring for details.

Examples:

```
# Make an operator acting like batch matrix A. Assume A.shape = [..., M, N]
operator = LinearOperator(...)
operator.shape = [..., M, N]

# Solve one linear system for every member of the batch.
RHS = ... # shape [..., M]

X = operator.solvevec(RHS)
# X is the solution to the linear system
# sum_j A[..., :, j] X[..., j] = RHS[..., :]

operator.matvec(X)
==> RHS
```

Args:

- **rhs**: **Tensor** with same **dtype** as this operator. **rhs** is treated like a [batch] vector meaning for every set of leading dimensions, the last dimension defines a vector. See class docstring for definition of compatibility regarding batch dimensions.
- **adjoint**: Python **bool**. If **True**, solve the system involving the adjoint of this **LinearOperator**: $A^H X = rhs$.
- **name**: A name scope to use for ops added by this method.

Returns:

Tensor with shape $[..., N]$ and same **dtype** as **rhs**.

Raises:

- **NotImplementedError**: If **self.is_non_singular** or **is_square** is False.

tensor_rank_tensor

```
tensor_rank_tensor(name='tensor_rank_tensor')
```

Rank (in the sense of tensors) of matrix corresponding to this operator.

If this operator acts like the batch matrix **A** with **A.shape** = $[B1, \dots, Bb, M, N]$, then this returns **b + 2**.

Args:

- **name**: A name for this `Op`.

Returns:

int32 Tensor, determined at runtime.

to_dense

```
to_dense(name='to_dense')
```

Return a dense (batch) matrix representing this operator.

trace

```
trace(name='trace')
```

Trace of the linear operator, equal to sum of `self.diag_part()` .

If the operator is square, this is also the sum of the eigenvalues.

Args:

- `name` : A name for this `Op` .

Returns:

Shape `[B1, ..., Bb]` `Tensor` of same `dtype` as `self` .

Except as otherwise noted, the content of this page is licensed under the [Creative Commons Attribution 3.0 License](#), and code samples are licensed under the [Apache 2.0 License](#). For details, see our [Site Policies](#). Java is a registered trademark of Oracle and/or its affiliates.

Last updated November 2, 2017.

Stay Connected

[Blog](#)

[GitHub](#)

[Twitter](#)

Support

[Issue Tracker](#)

[Release Notes](#)

[Stack Overflow](#)

English

[Terms](#) | [Privacy](#)