tf.contrib.distributions.bijectors.CholeskyOuterProduct

Contents

Class `CholeskyOuterProduct`

Inherits From: `Bijector`

Defined in `tensorflow/contrib/distributions/python/ops/bijectors/cholesky_outer_product_impl.py`.

See the guide: Random variable transformations (contrib) > Bijectors

Compute `g(X) = X @ X.T`; X is lower-triangular, positive-diagonal matrix.

`event_ndims` must be 0 or 2, i.e., scalar or matrix.

Note: the upper-triangular part of X is ignored (whether or not its zero).

The surjectivity of g as a map from the set of n x n positive-diagonal lower-triangular matrices to the set of SPD matrices follows immediately from executing the Cholesky factorization algorithm on an SPD matrix A to produce a positive-diagonal lower-triangular matrix L such that `A = L @ L.T`.

To prove the injectivity of g, suppose that L_1 and L_2 are lower-triangular with positive diagonals and satisfy `A = L_1 @ L_1.T = L_2 @ L_2.T`. Then `inv(L_1) @ A @ inv(L_1).T = [inv(L_1) @ L_2] @ [inv(L_1) @ L_2].T = I`. Setting `L_3 := inv(L_1) @ L_2`, that L_3 is a positive-diagonal lower-triangular matrix follows from `inv(L_1)` being positive-diagonal lower-triangular (which follows from the diagonal of a triangular matrix being its spectrum), and that the product of two positive-diagonal lower-triangular matrices is another positive-diagonal lower-triangular matrix.

A simple inductive argument (proceding one column of L_3 at a time) shows that, if `I = L_3 @ L_3.T`, with L_3 being lower-triangular with positive- diagonal, then `L_3 = I`. Thus, `L_1 = L_2`, proving injectivity of g.

Examples:

```
bijector.CholeskyOuterProduct(event_ndims=2).forward(x=[[1., 0], [2, 1]])
# Result: [[1., 2], [2, 5]], i.e., x @ x.T

bijector.CholeskyOuterProduct(event_ndims=2).inverse(y=[[1., 2], [2, 5]])
# Result: [[1., 0], [2, 1]], i.e., cholesky(y).
```

Properties

`dtype`

dtype of `Tensor` s transformable by this distribution.

`event_ndims`

Returns then number of event dimensions this bijector operates on.

`graph_parents`

Returns this `Bijector` 's graph_parents as a Python list.

`is_constant_jacobian`

Returns true iff the Jacobian is not a function of x.

> **Note:** Jacobian is either constant for both forward and inverse or neither.

Returns:

- `is_constant_jacobian` : Python `bool` .

`name`

Returns the string name of this `Bijector` .

`validate_args`

Returns True if Tensor arguments will be validated.

Methods

`__init__`

```
__init__(
    event_ndims=2,
    validate_args=False,
    name='cholesky_outer_product'
)
```

Instantiates the `CholeskyOuterProduct` bijector.

Args:

- `event_ndims` : `constant` `int32` scalar `Tensor` indicating the number of dimensions associated with a particular draw from the distribution. Must be 0 or 2.
- `validate_args` : Python `bool` indicating whether arguments should be checked for correctness.
- `name` : Python `str` name given to ops managed by this object.

Raises:

- `ValueError` : if event_ndims is neither 0 or 2.

`forward`

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

Returns the forward `Bijector` evaluation, i.e., X = g(Y).

Args:

- `x`: `Tensor`. The input to the "forward" evaluation.
- `name`: The name to give this op.

Returns:

`Tensor`.

Raises:

- `TypeError`: if `self.dtype` is specified and `x.dtype` is not `self.dtype`.
- `NotImplementedError`: if `_forward` is not implemented.

`forward_event_shape`

```
forward_event_shape(input_shape)
```

Shape of a single sample from a single batch as a `TensorShape`.

Same meaning as `forward_event_shape_tensor`. May be only partially defined.

Args:

- `input_shape`: `TensorShape` indicating event-portion shape passed into `forward` function.

Returns:

- `forward_event_shape_tensor`: `TensorShape` indicating event-portion shape after applying `forward`. Possibly unknown.

`forward_event_shape_tensor`

```
forward_event_shape_tensor(
    input_shape,
    name='forward_event_shape_tensor'
)
```

Shape of a single sample from a single batch as an `int32` 1D `Tensor`.

Args:

- `input_shape`: `Tensor`, `int32` vector indicating event-portion shape passed into `forward` function.

- `name`: name to give to the op

Returns:

- `forward_event_shape_tensor`: `Tensor`, `int32` vector indicating event-portion shape after applying `forward`.

`forward_log_det_jacobian`

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

Returns both the forward_log_det_jacobian.

Args:

- `x`: `Tensor`. The input to the "forward" Jacobian evaluation.
- `name`: The name to give this op.

Returns:

`Tensor`, if this bijector is injective. If not injective this is not implemented.

Raises:

- `TypeError`: if `self.dtype` is specified and `y.dtype` is not `self.dtype`.
- `NotImplementedError`: if neither `_forward_log_det_jacobian` nor {`_inverse`, `_inverse_log_det_jacobian`} are implemented, or this is a non-injective bijector.

`inverse`

```
inverse(
    y,
    name='inverse'
)
```

Returns the inverse `Bijector` evaluation, i.e., X = g^{-1}(Y).

Args:

- `y`: `Tensor`. The input to the "inverse" evaluation.
- `name`: The name to give this op.

Returns:

`Tensor`, if this bijector is injective. If not injective, returns the k-tuple containing the unique `k` points `(x1, ..., xk)` such that `g(xi) = y`.

Raises:

- `TypeError` : if `self.dtype` is specified and `y.dtype` is not `self.dtype` .
- `NotImplementedError` : if `_inverse` is not implemented.

`inverse_event_shape`

```
inverse_event_shape(output_shape)
```

Shape of a single sample from a single batch as a `TensorShape` .

Same meaning as `inverse_event_shape_tensor` . May be only partially defined.

Args:

- `output_shape` : `TensorShape` indicating event-portion shape passed into `inverse` function.

Returns:

- `inverse_event_shape_tensor` : `TensorShape` indicating event-portion shape after applying `inverse` . Possibly unknown.

`inverse_event_shape_tensor`

```
inverse_event_shape_tensor(
    output_shape,
    name='inverse_event_shape_tensor'
)
```

Shape of a single sample from a single batch as an `int32` 1D `Tensor` .

Args:

- `output_shape` : `Tensor` , `int32` vector indicating event-portion shape passed into `inverse` function.
- `name` : name to give to the op

Returns:

- `inverse_event_shape_tensor` : `Tensor` , `int32` vector indicating event-portion shape after applying `inverse` .

`inverse_log_det_jacobian`

```
inverse_log_det_jacobian(
    y,
    name='inverse_log_det_jacobian'
)
```

Returns the (log o det o Jacobian o inverse)(y).

Mathematically, returns: `log(det(dX/dY))(Y)` . (Recall that: `X=g^{-1}(Y)` .)

Note that `forward_log_det_jacobian` is the negative of this function, evaluated at `g^{-1}(y)` .

Args:

* `y`: `Tensor`. The input to the "inverse" Jacobian evaluation.
* `name`: The name to give this op.

Returns:

`Tensor`, if this bijector is injective. If not injective, returns the tuple of local log det Jacobians, `log(det(Dg_i^{-1}(y)))`, where `g_i` is the restriction of `g` to the `ith` partition `Di`.

Raises:

* `TypeError`: if `self.dtype` is specified and `y.dtype` is not `self.dtype`.
* `NotImplementedError`: if `_inverse_log_det_jacobian` is not implemented.

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.