

## tf.contrib.distributions.bijectors.Chain

## Contents

Class Chain

Properties

bijectors

dtype

Class **Chain**Inherits From: **Bijector**Defined in `tensorflow/contrib/distributions/python/ops/bijectors/chain_impl.py`.See the guide: [Random variable transformations \(contrib\) > Bijectors](#)

Bijector which applies a sequence of bijectors.

Example Use:

```
chain = Chain([Exp(), Softplus()], name="one_plus_exp")
```

Results in:

- Forward:

```
python exp = Exp() softplus = Softplus() Chain([exp, softplus]).forward(x) =  
exp.forward(softplus.forward(x)) = tf.exp(tf.log(1. + tf.exp(x))) = 1. + tf.exp(x)
```

- Inverse:

```
python exp = Exp() softplus = Softplus() Chain([exp, softplus]).inverse(y) =  
softplus.inverse(exp.inverse(y)) = tf.log(tf.exp(tf.log(y)) - 1.) = tf.log(y - 1.)
```

## Properties

**bijectors****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__(
    bijectors=None,
    validate_args=False,
    name=None
)
```

Instantiates **Chain** bijector.

Args:

- `bijectors`: Python **list** of bijector instances. An empty list makes this bijector equivalent to the **Identity** bijector.
- `validate_args`: Python **bool** indicating whether arguments should be checked for correctness.
- `name`: Python **str**, name given to ops managed by this object. Default: E.g., `Chain([Exp(), Softplus()]).name == "chain_of_exp_of_softplus"`.

Raises:

- **ValueError**: if bijectors have different dtypes.

## 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  $(\log \circ \det \circ \text{Jacobian} \circ \text{inverse})(y)$ .

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.

## Stay Connected

Blog

GitHub

Twitter

## Support

Issue Tracker

Release Notes

Stack Overflow

English

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