tfp.bijectors.Bijector

This API is new and only available via **`pip install tfp-nightly`**.

View source (https://github.com/tensorflow/probability/blob/master/tensorflow_probability/python/bij
on L1929)
GitHub

Interface for transformations of a `Distribution` sample.

```
@abc.abstractmethod
tfp.bijectors.Bijector(
    graph_parents=None, is_constant_jacobian=False, validate_args=False, dtype
    forward_min_event_ndims=UNSPECIFIED, inverse_min_event_ndims=UNSPECIFIED,
    experimental_use_kahan_sum=False, parameters=None, name=None
)
```

Bijectors can be used to represent any differentiable and injective (one to one) function defined on an open subset of `R^n`. Some non-injective transformations are also supported (see 'Non Injective Transforms' below).

Mathematical Details

A `Bijector` implements a smooth covering map (https://en.wikipedia.org/wiki/Local_diffeomorphism), i.e., a local diffeomorphism such that every point in the target has a neighborhood evenly covered by a map (see also (https://en.wikipedia.org/wiki/Covering_space#Covering_of_a_manifold)). A `Bijector` is used by `TransformedDistribution` but can be generally used for transforming a `Distribution` generated `Tensor`. A `Bijector` is characterized by three operations:

1. Forward

   Useful for turning one random outcome into another random outcome from a different distribution.

2. Inverse

Useful for 'reversing' a transformation to compute one probability in terms of another.

3. **`log_det_jacobian(x)`**

   'The log of the absolute value of the determinant of the matrix of all first-order partial derivatives of the inverse function.'

   Useful for inverting a transformation to compute one probability in terms of another. Geometrically, the Jacobian determinant is the volume of the transformation and is used to scale the probability.

   We take the absolute value of the determinant before log to avoid NaN values. Geometrically, a negative determinant corresponds to an orientation-reversing transformation. It is ok for us to discard the sign of the determinant because we only integrate everywhere-nonnegative functions (probability densities) and the correct orientation is always the one that produces a nonnegative integrand.

By convention, transformations of random variables are named in terms of the forward transformation. The forward transformation creates samples, the inverse is useful for computing probabilities.

Example Uses

- Basic properties:

```
x = ...  # A tensor.
# Evaluate forward transformation.
fwd_x = my_bijector.forward(x)
x == my_bijector.inverse(fwd_x)
x != my_bijector.forward(fwd_x)  # Not equal because x != g(g(x)).
```

- Computing a log-likelihood:

```
def transformed_log_prob(bijector, log_prob, x):
  return (bijector.inverse_log_det_jacobian(x, event_ndims=0) +
          log_prob(bijector.inverse(x)))
```

- Transforming a random outcome:

```
def transformed_sample(bijector, x):
  return bijector.forward(x)
```

Example Bijectors

- 'Exponential'

```
Y = g(X) = exp(X)
X ~ Normal(0, 1)  # Univariate.
```

Implies:

```
g^{-1}(Y) = log(Y)
|Jacobian(g^{-1})(y)| = 1 / y
Y ~ LogNormal(0, 1), i.e.,
prob(Y=y) = |Jacobian(g^{-1})(y)| * prob(X=g^{-1}(y))
          = (1 / y) Normal(log(y); 0, 1)
```

Here is an example of how one might implement the `Exp` bijector:

```
class Exp(Bijector):

  def __init__(self, validate_args=False, name='exp'):
    super(Exp, self).__init__(
        validate_args=validate_args,
        forward_min_event_ndims=0,
        name=name)

  def _forward(self, x):
    return tf.exp(x)

  def _inverse(self, y):
    return tf.log(y)

  def _inverse_log_det_jacobian(self, y):
    return -self._forward_log_det_jacobian(self._inverse(y))

  def _forward_log_det_jacobian(self, x):
    # Notice that we needn't do any reducing, even when`event_ndims > 0
    # The base Bijector class will handle reducing for us; it knows how
    # to do so because we called `super` `__init__` with
    # `forward_min_event_ndims = 0`.
    return x
```
```
```

- 'ScaleMatvecTriL'

```
Y = g(X) = sqrtSigma * X
X ~ MultivariateNormal(0, I_d)
```

Implies:

```
g^{-1}(Y) = inv(sqrtSigma) * Y
|Jacobian(g^{-1})(y)| = det(inv(sqrtSigma))
Y ~ MultivariateNormal(0, sqrtSigma) , i.e.,
prob(Y=y) = |Jacobian(g^{-1})(y)| * prob(X=g^{-1}(y))
          = det(sqrtSigma)^(-d) *
            MultivariateNormal(inv(sqrtSigma) * y; 0, I_d)
```
```

#### Min_event_ndims and Naming

Bijectors are named for the dimensionality of data they act on (i.e. without broadcasting). We can think of bijectors having an intrinsic `min_event_ndims` , which is the minimum number of dimensions for the bijector act on. For instance, a Cholesky decomposition requires a matrix, and hence `min_event_ndims=2`.

#### Some examples:

**`Cholesky: min_event_ndims=2`** **`Exp: min_event_ndims=0`** **`MatvecTriL: min_event_ndims=1`** **`Scale: min_event_ndims=0`** **`Sigmoid: min_event_ndims=0`** **`SoftmaxCentered: min_event_ndims=1`**

For multiplicative transformations, note that `Scale` operates on scalar events, whereas the `Matvec*` bijectors operate on vector-valued events.

More generally, there is a `forward_min_event_ndims` and an `inverse_min_event_ndims`. In most cases, these will be the same. However, for some shape changing bijectors, these will be different (e.g. a bijector which pads an extra dimension at the end, might have `forward_min_event_ndims=0` and `inverse_min_event_ndims=1`.

##### Additional Considerations for "Multi Tensor" Bijectors
```

Bijectors which operate on structures of **`Tensor`** require structured **`min_event_ndims`** matching the structure of the inputs. In these cases, **`min_event_ndims`** describes both the minimum dimensionality *and* the structure of arguments to **`forward`** and **`inverse`**. For example:

```
Split([sizes], axis):
  forward_min_event_ndims=-axis
  inverse_min_event_ndims=[-axis] * len(sizes)
```

3y default, we require **`shape(x[i])[-event_ndims:-min_event_ndims]`** to be identical for all elements i
uctured input **`x`**. Specifically, broadcasting over non-minimal event-dims is generally not allowed for struct
, with the exception described in the next paragraph.

Independent parts: multipart transformations in which the parts do not interact with each other, such as **`tfd.JointMap`**, **`tfd.Restructure`**, and chains of these, may allow **`event_ndims[i] - min_event_ndims[i]`** to take different values across different parts. The parts must still share a common (broadcast) batch shape---the shape of the log Jacobian determinant--- but independence removes the requirement for further alignment in the event shapes. For example, a **`JointMap`** bijector may be used to transform distributions of varying event rank and size, even when other multipart bijectors such as **`tfb.Invert(tfb.Split(n))`** would require all inputs to have the same event rank:

```
jm = tfb.JointMap([tfb.Scale([1., 2.],
                  tfb.Scale([3., 4., 5.]))])

fldj = jm.forward_log_det_jacobian([tf.ones([2]), tf.ones([3])],
                                  event_ndims=[1, 1])
# ==> `fldj` has shape `[]`.

fldj = jm.forward_log_det_jacobian([tf.ones([2]), tf.ones([3])],
                                  event_ndims=[1, 0])
# ==> `fldj` has shape `[3]` (the shape-`[2]` input part is implicitly
#      broadcast to shape `[3, 2]`, creating a common batch shape).

fldj = jm.forward_log_det_jacobian([tf.ones([2]), tf.ones([3])],
                                  event_ndims=[0, 0])
# ==> Error; `[2]` and `[3]` do not broadcast to a consistent batch shape.
```

Jacobian Determinant

The Jacobian determinant of a single-part bijector is a reduction over `event_ndims - min_event_ndims` (`forward_min_event_ndims` for `forward_log_det_jacobian` and `inverse_min_event_ndims` for `inverse_log_det_jacobian`).

To see this, consider the `Exp Bijector` applied to a `Tensor` which has sample, batch, and event (S, B, E) shape semantics. Suppose the `Tensor`'s partitioned-shape is `(S=[4], B=[2], E=[3, 3])`. The shape of the `Tensor` returned by `forward` and `inverse` is unchanged, i.e., `[4, 2, 3, 3]`. However the shape returned by `inverse_log_det_jacobian` is `[4, 2]` because the Jacobian determinant is a reduction over the event dimensions.

Another example is the `ScaleMatvecDiag Bijector`. Because `min_event_ndims = 1`, the Jacobian determinant reduction is over `event_ndims - 1`.

It is sometimes useful to implement the inverse Jacobian determinant as the negative forward Jacobian determinant. For example,

```
def _inverse_log_det_jacobian(self, y):
   return -self._forward_log_det_jac(self._inverse(y))  # Note negation.
```

The correctness of this approach can be seen from the following claim.

- Claim:

  Assume `Y = g(X)` is a bijection whose derivative exists and is nonzero for its domain, i.e., `dY/dX = d/dX g(X) != 0`. Then:

  ```
  (log o det o jacobian o g^{-1})(Y) = -(log o det o jacobian o g)(X)
  ```

- Proof:

  From the bijective, nonzero differentiability of `g`, the inverse function theorem (https://en.wikipedia.org/wiki/Inverse_function_theorem) implies `g^{-1}` is differentiable in the image of `g`. Applying the chain rule to `y = g(x) = g(g^{-1}(y))` yields `I = g'(g^{-1}(y))*g^{-1}'(y)`. The same theorem also implies `g^{-1}'` is non-singular therefore: `inv[ g'(g^{-1}(y)) ] = g^{-1}'(y)`. The claim follows from properties of determinant (https://en.wikipedia.org/wiki/Determinant#Multiplicativity_and_matrix_groups).

Generally it's preferable to directly implement the inverse Jacobian determinant. This should have superior numerical stability and will often share subgraphs with the `_inverse` implementation.

Note that Jacobian determinants are always a single Tensor (potentially with batch dimensions), even for bijectors that act on multipart structures, since any multipart transformation may be viewed as a transformation on a single (possibly batched) vector obtained by flattening and concatenating the input parts.

Is_constant_jacobian

Certain bijectors will have constant jacobian matrices. For instance, the `ScaleMatvecTriL` bijector encodes multiplication by a lower triangular matrix, with jacobian matrix equal to the same aforementioned matrix.

`is_constant_jacobian` encodes the fact that the jacobian matrix is constant. The semantics of this argument are the following:

- Repeated calls to 'log_det_jacobian' functions with the same `event_ndims` (but not necessarily same input), will return the first computed jacobian (because the matrix is constant, and hence is input independent).
- `log_det_jacobian` implementations are merely broadcastable to the true `log_det_jacobian` (because, again, the jacobian matrix is input independent). Specifically, `log_det_jacobian` is implemented as the log jacobian determinant for a single input.

```
class Identity(Bijector):

  def __init__(self, validate_args=False, name='identity'):
    super(Identity, self).__init__(
        is_constant_jacobian=True,
        validate_args=validate_args,
        forward_min_event_ndims=0,
        name=name)

  def _forward(self, x):
    return x

  def _inverse(self, y):
    return y

  def _inverse_log_det_jacobian(self, y):
    return -self._forward_log_det_jacobian(self._inverse(y))
```

```
def _forward_log_det_jacobian(self, x):
  # The full log jacobian determinant would be tf.zero_like(x).
  # However, we circumvent materializing that, since the jacobian
  # calculation is input independent, and we specify it for one input.
  return tf.constant(0., x.dtype)
```

Subclass Requirements

- Subclasses typically implement:
  - **`_forward`**,
  - **`_inverse`**,
  - **`_inverse_log_det_jacobian`**,
  - **`_forward_log_det_jacobian`** (optional),
  - **`_is_increasing`** (scalar bijectors only)

  The **`_forward_log_det_jacobian`** is called when the bijector is inverted via the **`Invert`** bijector. If undefined, a slightly less efficiently calculation, **`-1 * _inverse_log_det_jacobian`**, is used.

  If the bijector changes the shape of the input, you must also implement:
  - _forward_event_shape_tensor,
  - _forward_event_shape (optional),
  - _inverse_event_shape_tensor,
  - _inverse_event_shape (optional).

  By default the event-shape is assumed unchanged from input.

  Multipart bijectors, which operate on structures of tensors, may implement additional methods to propogate calltime dtype information over any changes to structure. These methods are:
  - _forward_dtype
  - _inverse_dtype
  - _forward_event_ndims
  - _inverse_event_ndims

- If the `Bijector`'s use is limited to `TransformedDistribution` (or friends like `QuantizedDistribution`) then depending on your use, you may not need to implement all of `_forward` and `_inverse` functions.

  Examples:

  1. Sampling (e.g., `sample`) only requires `_forward`.
  2. Probability functions (e.g., `prob`, `cdf`, `survival`) only require `_inverse` (and related).
  3. Only calling probability functions on the output of `sample` means `_inverse` can be implemented as a cache lookup.

  See 'Example Uses' [above] which shows how these functions are used to transform a distribution. (Note: `_forward` could theoretically be implemented as a cache lookup but this would require controlling the underlying sample generation mechanism.)

Non Injective Transforms

ıg: Handling of non-injective transforms is subject to change.

Non injective maps `g` are supported, provided their domain `D` can be partitioned into `k` disjoint subsets, `Union{D1, ..., Dk}`, such that, ignoring sets of measure zero, the restriction of `g` to each subset is a differentiable bijection onto `g(D)`. In particular, this implies that for `y in g(D)`, the set inverse, i.e. `g^{-1}(y) = {x in D : g(x) = y}`, always contains exactly `k` distinct points.

The property, `_is_injective` is set to `False` to indicate that the bijector is not injective, yet satisfies the above condition.

The usual bijector API is modified in the case `_is_injective is False` (see method docstrings for specifics). Here we show by example the `AbsoluteValue` bijector. In this case, the domain `D = (-inf, inf)`, can be partitioned into `D1 = (-inf, 0)`, `D2 = {0}`, and `D3 = (0, inf)`. Let `gi` be the restriction of `g` to `Di`, then both `g1` and `g3` are bijections onto `(0, inf)`, with `g1^{-1}(y) = -y`, and `g3^{-1}(y) = y`. We will use `g1` and `g3` to define bijector methods over `D1` and `D3`. `D2 = {0}` is an oddball in that `g2` is one to one, and the derivative is not well defined. Fortunately, when considering transformations of probability densities (e.g. in `TransformedDistribution`), sets of measure zero have no effect in theory, and only a small effect in 32 or 64 bit precision. For that reason, we define `inverse(0)` and `inverse_log_det_jacobian(0)` both as `[0, 0]`, which is convenient and results in a left-semicontinuous pdf.

```
abs = tfp.bijectors.AbsoluteValue()

abs.forward(-1.)
==> 1.

abs.forward(1.)
==> 1.

abs.inverse(1.)
==> (-1., 1.)

# The |dX/dY| is constant, == 1.  So Log|dX/dY| == 0.
abs.inverse_log_det_jacobian(1., event_ndims=0)
==> (0., 0.)

# Special case handling of 0.
abs.inverse(0.)
==> (0., 0.)

abs.inverse_log_det_jacobian(0., event_ndims=0)
==> (0., 0.)
```

Args	
`graph_parents`	Python list of graph prerequisites of this `Bijector`.
`is_constant_jacobian`	Python `bool` indicating that the Jacobian matrix is not a function of the input.
`validate_args`	Python `bool`, default `False`. Whether to validate input with asserts. If `validate_args` is `False`, and the inputs are invalid, correct behavior is not guaranteed.
`dtype`	`tf.dtype` supported by this `Bijector`. `None` means dtype is not enforced. For multipart bijectors, this value is expected to be the same for all elements of the input and output structures.
`forward_min_event_ndims`	Python `integer` (structure) indicating the minimum number of dimensions on which `forward` operates.
`inverse_min_event_ndims`	Python `integer` (structure) indicating the minimum number of dimensions on which `inverse` operates. Will be set to `forward_min_event_ndims` by default, if no value is provided.
`experimental_use_kahan_sum`	Python `bool`. When `True`, use Kahan summation to aggregate log-det jacobians from independent underlying log-det jacobian values, which improves against the precision of a naive float32 sum. This can be

	noticeable in particular for large dimensions in float32. See CPU caveat on **`tfp.math.reduce_kahan_sum`** (https://www.tensorflow.org/probability/api_docs/python/tfp/math/reduce_kahan_sum) .
`parameters`	Python `dict` of parameters used to instantiate this `Bijector`. Bijector instances with identical types, names, and `parameters` share an input/output cache. `parameters` dicts are keyed by strings and are identical if their keys are identical and if corresponding values have identical hashes (or object ids, for unhashable objects).
`name`	The name to give Ops created by the initializer.

Raises	
`ValueError`	If neither `forward_min_event_ndims` and `inverse_min_event_ndims` are specified, or if either of them is negative.
`ValueError`	If a member of `graph_parents` is not a `Tensor`.

Attributes	
`dtype`	
`forward_min_event_ndims`	Returns the minimal number of dimensions bijector.forward operates on. Multipart bijectors return structured `ndims`, which indicates the expected structure of their inputs. Some multipart bijectors, notably Composites, may return structures of `None`.
`graph_parents`	Returns this `Bijector`'s graph_parents as a Python list.
`has_static_min_event_ndims`	Returns True if the bijector has statically-known `min_event_ndims`. (deprecated) **Warning:** THIS FUNCTION IS DEPRECATED. It will be removed after 2021-08-01. Instructions for updating: `min_event_ndims` is now static for all bijectors; this property is no longer needed.

`inverse_min_event_ndims`	Returns the minimal number of dimensions bijector.inverse operates on. Multipart bijectors return structured `event_ndims`, which indicates the expected structure of their outputs. Some multipart bijectors, notably Composites, may return structures of `None`.
`is_constant_jacobian`	Returns true iff the Jacobian matrix is not a function of x. **Note:** Jacobian matrix is either constant for both forward and inverse or neither.
`name`	Returns the string name of this `Bijector`.
`name_scope`	Returns a `tf.name_scope` (https://www.tensorflow.org/api_docs/python/tf/name_scope) instance for this class.
`non_trainable_variables`	Sequence of non-trainable variables owned by this module and its submodules. **Note:** this method uses reflection to find variables on the current instance and submodules. For performance reasons you may wish to cache the result of calling this method if you don't expect the return value to change.
`parameters`	Dictionary of parameters used to instantiate this `Bijector`.

`submodules`	Sequence of all sub-modules.
	Submodules are modules which are properties of this module, or found as properties of modules which are properties of this module (and so on).
	
	`>>> a = tf.Module()`
	`>>> b = tf.Module()`
	`>>> c = tf.Module()`
	`>>> a.b = b`
	`>>> b.c = c`
	`>>> list(a.submodules) == [b, c]`
	`True`
	`>>> list(b.submodules) == [c]`
	`True`
	`>>> list(c.submodules) == []`
	`True`

`trainable_variables`	Sequence of trainable variables owned by this module and its submodules. **Note:** this method uses reflection to find variables on the current instance and submodules. For performance reasons you may wish to cache the result of calling this method if you don't expect the return value to change.
`validate_args`	Returns True if Tensor arguments will be validated.
`variables`	Sequence of variables owned by this module and its submodules. **Note:** this method uses reflection to find variables on the current instance and submodules. For performance reasons you may wish to cache the result of calling this method if you don't expect the return value to change.

Attributes

`dtype`	
`forward_min_event_ndims`	Returns the minimal number of dimensions bijector.forward operates on. Multipart bijectors return structured `ndims`, which indicates the expected structure of their inputs. Some multipart bijectors, notably Composites, may return structures of `None`.
`graph_parents`	Returns this `Bijector`'s graph_parents as a Python list.
`has_static_min_event_ndims`	Returns True if the bijector has statically-known `min_event_ndims`. (deprecated) **Warning:** THIS FUNCTION IS DEPRECATED. It will be removed after 2021-08-01. Instructions for updating: `min_event_ndims` is now static for all bijectors; this property is no longer needed.
`inverse_min_event_ndims`	Returns the minimal number of dimensions bijector.inverse operates on.

	Multipart bijectors return structured `event_ndims`, which indicates the expected structure of their outputs. Some multipart bijectors, notably Composites, may return structures of `None`.
`is_constant_jacobian`	Returns true iff the Jacobian matrix is not a function of x. **Note:** Jacobian matrix is either constant for both forward and inverse or neither.
`name`	Returns the string name of this `Bijector`.
`name_scope`	Returns a [`tf.name_scope`](https://www.tensorflow.org/api_docs/python/tf/name_scope) (https://www.tensorflow.org/api_docs/python/tf/name_scope) instance for this class.
`non_trainable_variables`	Sequence of non-trainable variables owned by this module and its submodules. **Note:** this method uses reflection to find variables on the current instance and submodules. For performance reasons you may wish to cache the result of calling this method if you don't expect the return value to change.
`parameters`	Dictionary of parameters used to instantiate this `Bijector`.

`submodules`	Sequence of all sub-modules.
	Submodules are modules which are properties of this module, or found as properties of modules which are properties of this module (and so on).
	
	`>>> a = tf.Module()`
	`>>> b = tf.Module()`
	`>>> c = tf.Module()`
	`>>> a.b = b`
	`>>> b.c = c`
	`>>> list(a.submodules) == [b, c]`
	`True`
	`>>> list(b.submodules) == [c]`
	`True`
	`>>> list(c.submodules) == []`
	`True`

`trainable_variables`	Sequence of trainable variables owned by this module and its

	submodules. **Note:** this method uses reflection to find variables on the current instance and submodules. For performance reasons you may wish to cache the result of calling this method if you don't expect the return value to change.
`validate_args`	Returns True if Tensor arguments will be validated.
`variables`	Sequence of variables owned by this module and its submodules. **Note:** this method uses reflection to find variables on the current instance and submodules. For performance reasons you may wish to cache the result of calling this method if you don't expect the return value to change.

Methods

`copy`

View source
(https://github.com/tensorflow/probability/blob/master/tensorflow_probability/python/bijectors/bijector.py#L959-L979)

```
copy(
    **override_parameters_kwargs
)
```

Creates a copy of the bijector.

the copy bijector may continue to depend on the original initialization arguments.

Args	
`**override_parameters_kwargs`	String/value dictionary of initialization arguments to override with new values.

Returns

`bijector`	A new instance of `type(self)` initialized from the union of self.parameters and override_parameters_kwargs, i.e., `dict(self.parameters, **override_parameters_kwargs)`.

`experimental_batch_shape`

View source
(https://github.com/tensorflow/probability/blob/master/tensorflow_probability/python/bijectors/bijector.py#L1153-L1203)

```
experimental_batch_shape(
    x_event_ndims=None, y_event_ndims=None
)
```

Returns the batch shape of this bijector for inputs of the given rank.

The batch shape of a bijector decribes the set of distinct transformations it represents on events of a given size. For example: the bijector `tfb.Scale([1., 2.])` has batch shape `[2]` for scalar events (`event_ndims = 0`), because applying it to a scalar event produces two scalar outputs, the result of two different scaling transformations. The same bijector has batch shape `[]` for vector events, because applying it to a vector produces (via elementwise multiplication) a single vector output.

Bijectors that operate independently on multiple state parts, such as `tfb.JointMap`, must broadcast to a coherent batch shape. Some events may not be valid: for example, the bijector `tfd.JointMap([tfb.Scale([1., 2.]), tfb.Scale([1., 2., 3.])])` does not produce a valid batch shape when `event_ndims = [0, 0]`, since the batch shapes of the two parts are inconsistent. The same bijector does define valid batch shapes of `[]`, `[2]`, and `[3]` if `event_ndims` is `[1, 1]`, `[0, 1]`, or `[1, 0]`, respectively.

Since transforming a single event produces a scalar log-det-Jacobian, the batch shape of a bijector with non-constant Jacobian is expected to equal the shape of `forward_log_det_jacobian(x, event_ndims=x_event_ndims)` or `inverse_log_det_jacobian(y, event_ndims=y_event_ndims)`, for `x` or `y` of the specified `ndims`.

Args	
`x_event_ndims`	Optional Python `int` (structure) number of dimensions in a probabilistic event passed to `forward`; this must be greater than or equal to `self.forward_min_event_ndims`. If `None`, defaults to

	`self.forward_min_event_ndims`. Mutually exclusive with `y_event_ndims`. Default value: `None`.
`y_event_ndims`	Optional Python `int` (structure) number of dimensions in a probabilistic event passed to `inverse`; this must be greater than or equal to `self.inverse_min_event_ndims`. Mutually exclusive with `x_event_ndims`. Default value: `None`.

Returns	
`batch_shape`	`TensorShape` batch shape of this bijector for a value with the given event rank. May be unknown or partially defined.

`experimental_batch_shape_tensor`

View source (https://github.com/tensorflow/probability/blob/master/tensorflow_probability/python/bijectors/bijector.py#L1219-L1270)

```
experimental_batch_shape_tensor(
    x_event_ndims=None, y_event_ndims=None
)
```

Returns the batch shape of this bijector for inputs of the given rank.

The batch shape of a bijector decribes the set of distinct transformations it represents on events of a given size. For example: the bijector `tfb.Scale([1., 2.])` has batch shape `[2]` for scalar events (`event_ndims = 0`), because applying it to a scalar event produces two scalar outputs, the result of two different scaling transformations. The same bijector has batch shape `[]` for vector events, because applying it to a vector produces (via elementwise multiplication) a single vector output.

Bijectors that operate independently on multiple state parts, such as `tfb.JointMap`, must broadcast to a coherent batch shape. Some events may not be valid: for example, the bijector `tfd.JointMap([tfb.Scale([1., 2.]), tfb.Scale([1., 2., 3.])])` does not produce a valid batch shape when `event_ndims = [0, 0]`, since the batch shapes of the two parts are inconsistent. The same bijector does define valid batch shapes of `[]`, `[2]`, and `[3]` if `event_ndims` is `[1, 1]`, `[0, 1]`, or `[1, 0]`, respectively.

Since transforming a single event produces a scalar log-det-Jacobian, the batch shape of a bijector with non-constant Jacobian is expected to equal the shape of `forward_log_det_jacobian(x, event_ndims=x_event_ndims)` or `inverse_log_det_jacobian(y, event_ndims=y_event_ndims)`, for `x` or `y` of the specified `ndims`.

Args	
`x_event_ndims`	Optional Python `int` (structure) number of dimensions in a probabilistic event passed to `forward`; this must be greater than or equal to `self.forward_min_event_ndims`. If `None`, defaults to `self.forward_min_event_ndims`. Mutually exclusive with `y_event_ndims`. Default value: `None`.
`y_event_ndims`	Optional Python `int` (structure) number of dimensions in a probabilistic event passed to `inverse`; this must be greater than or equal to `self.inverse_min_event_ndims`. Mutually exclusive with `x_event_ndims`. Default value: `None`.

Returns	
`batch_shape_tensor`	integer `Tensor` batch shape of this bijector for a value with the given event rank.

`forward`

View source
(https://github.com/tensorflow/probability/blob/master/tensorflow_probability/python/bijectors/bijector.py#L1330-L1346)

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

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

Args	
`x`	`Tensor` (structure). The input to the 'forward' evaluation.
`name`	The name to give this op.
`**kwargs`	Named arguments forwarded to subclass implementation.

Returns
`Tensor` (structure).

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

`forward_dtype`

View source
(https://github.com/tensorflow/probability/blob/master/tensorflow_probability/python/bijectors/bijector.py#L1670-L1697)

```
forward_dtype(
    dtype=UNSPECIFIED, name='forward_dtype', **kwargs
)
```

Returns the dtype returned by `forward` for the provided input.

`forward_event_ndims`

View source
(https://github.com/tensorflow/probability/blob/master/tensorflow_probability/python/bijectors/bijector.py#L1728-L1755)

```
forward_event_ndims(
    event_ndims, **kwargs
)
```

Returns the number of event dimensions produced by `forward`.

Args	
`event_ndims`	Structure of Python and/or Tensor `int`s, and/or `None` values. The structure should match that of `self.forward_min_event_ndims`, and all non-`None` values must be greater than or equal to the corresponding value in `self.forward_min_event_ndims`.

`**kwargs`	Optional keyword arguments forwarded to nested bijectors.

Returns	
`forward_event_ndims`	Structure of integers and/or **`None`** values matching **`self.inverse_min_event_ndims`**. These are computed using 'prefer static' semantics: if any inputs are **`None`**, some or all of the outputs may be **`None`**, indicating that the output dimension could not be inferred (conversely, if all inputs are non-**`None`**, all outputs will be non-**`None`**). If all input **`event_ndims`** are Python **`int`**s, all of the (non-**`None`**) outputs will be Python **`int`**s; otherwise, some or all of the outputs may be **`Tensor`** **`int`**s.

`forward_event_shape`

View source (https://github.com/tensorflow/probability/blob/master/tensorflow_probability/python/bijectors/bijector.py#L1023-L1043)

```
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`** (structure) indicating event-portion shape passed into **`forward`** function.

Returns	
`forward_event_shape_tensor`	**`TensorShape`** (structure) indicating event-portion shape after applying **`forward`**. Possibly unknown.

`forward_event_shape_tensor`

View source
(https://github.com/tensorflow/probability/blob/master/tensorflow_probability/python/bijectors/bijector.py#L986-L1016)

```
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 (structure) indicating event-portion shape passed into `forward` function.
`name`	name to give to the op

Returns	
`forward_event_shape_tensor`	`Tensor`, `int32` vector (structure) indicating event-portion shape after applying `forward`.

`forward_log_det_jacobian`

View source
(https://github.com/tensorflow/probability/blob/master/tensorflow_probability/python/bijectors/bijector.py#L1623-L1660)

```
forward_log_det_jacobian(
    x, event_ndims=None, name='forward_log_det_jacobian', **kwargs
)
```

Returns both the forward_log_det_jacobian.

Args	
`x`	`Tensor` (structure). The input to the 'forward' Jacobian determinant evaluation.
`event_ndims`	Optional number of dimensions in the probabilistic events being

	transformed; this must be greater than or equal to **`self.forward_min_event_ndims`**. If **`event_ndims`** is specified, the log Jacobian determinant is summed to produce a scalar log-determinant for each event. Otherwise (if **`event_ndims`** is **`None`**), no reduction is performed. Multipart bijectors require *structured* event_ndims, such that the batch rank **`rank(y[i]) - event_ndims[i]`** is the same for all elements **`i`** of the structured input. In most cases (with the exception of **`tfb.JointMap`**) they further require that **`event_ndims[i] - self.inverse_min_event_ndims[i]`** is the same for all elements **`i`** of the structured input. Default value: **`None`** (equivalent to **`self.forward_min_event_ndims`**).
`name`	The name to give this op.
`**kwargs`	Named arguments forwarded to subclass implementation.

Returns
`Tensor` (structure), if this bijector is injective. If not injective this is not implemented.

Raises	
`TypeError`	if **`y`**'s dtype is incompatible with the expected output dtype.
`NotImplementedError`	if neither **`_forward_log_det_jacobian`** nor {**`_inverse`**, **`_inverse_log_det_jacobian`**} are implemented, or this is a non-injective bijector.
`ValueError`	if the value of **`event_ndims`** is not valid for this bijector.

`inverse`

View source (https://github.com/tensorflow/probability/blob/master/tensorflow_probability/python/bijectors/bijector.py#L1391-L1409)

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

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

Args	
`y`	`Tensor` (structure). The input to the 'inverse' evaluation.
`name`	The name to give this op.
`**kwargs`	Named arguments forwarded to subclass implementation.

Returns
`Tensor` (structure), 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 `y`'s structured dtype is incompatible with the expected output dtype.
`NotImplementedError`	if `_inverse` is not implemented.

`inverse_dtype`

View source
 (https://github.com/tensorflow/probability/blob/master/tensorflow_probability/python/bijectors/bijector.py#L1699-L1726)

```
inverse_dtype(
    dtype=UNSPECIFIED, name='inverse_dtype', **kwargs
)
```

Returns the dtype returned by `inverse` for the provided input.

`inverse_event_ndims`

View source
 (https://github.com/tensorflow/probability/blob/master/tensorflow_probability/python/bijectors/bijector.py#L1757-L1784)

```
inverse_event_ndims(
    event_ndims, **kwargs
)
```

Returns the number of event dimensions produced by `inverse`.

Args	
`event_ndims`	Structure of Python and/or Tensor `int`s, and/or `None` values. The structure should match that of `self.inverse_min_event_ndims`, and all non-`None` values must be greater than or equal to the corresponding value in `self.inverse_min_event_ndims`.
`**kwargs`	Optional keyword arguments forwarded to nested bijectors.

Returns	
`inverse_event_ndims`	Structure of integers and/or `None` values matching `self.forward_min_event_ndims`. These are computed using 'prefer static' semantics: if any inputs are `None`, some or all of the outputs may be `None`, indicating that the output dimension could not be inferred (conversely, if all inputs are non-`None`, all outputs will be non-`None`). If all input `event_ndims` are Python `int`s, all of the (non-`None`) outputs will be Python `int`s; otherwise, some or all of the outputs may be `Tensor` `int`s.

`inverse_event_shape`

View source
 (https://github.com/tensorflow/probability/blob/master/tensorflow_probability/python/bijectors/bijector.py#L1085-L1105)

```
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` (structure) indicating event-portion shape passed into `inverse` function.

Returns

`inverse_event_shape_tensor`	**`TensorShape`** (structure) indicating event-portion shape after applying **`inverse`**. Possibly unknown.

`inverse_event_shape_tensor`

View source
(https://github.com/tensorflow/probability/blob/master/tensorflow_probability/python/bijectors/bijector.py#L1050-L1078)

```
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 (structure) indicating event-portion shape passed into **`inverse`** function.
`name`	name to give to the op

Returns	
`inverse_event_shape_tensor`	**`Tensor`**, **`int32`** vector (structure) indicating event-portion shape after applying **`inverse`**.

`inverse_log_det_jacobian`

View source
(https://github.com/tensorflow/probability/blob/master/tensorflow_probability/python/bijectors/bijector.py#L1497-L1539)

```
inverse_log_det_jacobian(
    y, event_ndims=None, name='inverse_log_det_jacobian', **kwargs
)
```

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` (structure). The input to the 'inverse' Jacobian determinant evaluation.
`event_ndims`	Optional number of dimensions in the probabilistic events being transformed; this must be greater than or equal to `self.inverse_min_event_ndims`. If `event_ndims` is specified, the log Jacobian determinant is summed to produce a scalar log-determinant for each event. Otherwise (if `event_ndims` is `None`), no reduction is performed. Multipart bijectors require *structured* event_ndims, such that the batch rank `rank(y[i]) - event_ndims[i]` is the same for all elements `i` of the structured input. In most cases (with the exception of `tfb.JointMap`) they further require that `event_ndims[i] - self.inverse_min_event_ndims[i]` is the same for all elements `i` of the structured input. Default value: `None` (equivalent to `self.inverse_min_event_ndims`).
`name`	The name to give this op.
`**kwargs`	Named arguments forwarded to subclass implementation.

Returns	
`ildj`	`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 `x`'s dtype is incompatible with the expected inverse-dtype.
`NotImplementedError`	if `_inverse_log_det_jacobian` is not implemented.
`ValueError`	if the value of `event_ndims` is not valid for this bijector.

`parameter_properties`

View source (https://github.com/tensorflow/probability/blob/master/tensorflow_probability/python/bijectors/bijector.py#L1277-L1296)

```
@classmethod
parameter_properties(
    dtype=tf.float32
)
```

Returns a dict mapping constructor arg names to property annotations.

This dict should include an entry for each of the bijector's `Tensor`-valued constructor arguments.

Args	
`dtype`	Optional float `dtype` to assume for continuous-valued parameters. Some constraining bijectors require advance knowledge of the dtype because certain constants (e.g., `tfb.Softplus.low`) must be instantiated with the same dtype as the values to be transformed.

Returns	
`parameter_properties`	A **`str -`** **>**tfp.python.internal.parameter_properties.ParameterProperties**`dict mapping constructor argument names to`**ParameterProperties` instances.

`with_name_scope`

```
@classmethod
with_name_scope(
    method
)
```

Decorator to automatically enter the module name scope.

```
>>> class MyModule(tf.Module):
...   @tf.Module.with_name_scope
```

```
...   def __call__(self, x):
...     if not hasattr(self, 'w'):
...       self.w = tf.Variable(tf.random.normal([x.shape[1], 3]))
...     return tf.matmul(x, self.w)
```

Using the above module would produce **`tf.Variable`** (https://www.tensorflow.org/api_docs/python/tf/Variable)s and **`tf.Tensor`** (https://www.tensorflow.org/api_docs/python/tf/Tensor)s whose names included the module name:

```
>>> mod = MyModule()
>>> mod(tf.ones([1, 2]))
<tf.Tensor: shape=(1, 3), dtype=float32, numpy=..., dtype=float32)>
>>> mod.w
<tf.Variable 'my_module/Variable:0' shape=(2, 3) dtype=float32,
numpy=..., dtype=float32)>
```

Args	
`method`	The method to wrap.

Returns
The original method wrapped such that it enters the module's name scope.

`__call__`

View source (https://github.com/tensorflow/probability/blob/master/tensorflow_probability/python/bijectors/bijector.py#L872-L957)

```
__call__(
    value, name=None, **kwargs
)
```

Applies or composes the `Bijector`, depending on input type.

This is a convenience function which applies the `Bijector` instance in three different ways, depending on the input:

1. If the input is a `tfd.Distribution` instance, return `tfd.TransformedDistribution(distribution=input, bijector=self)`.
2. If the input is a `tfb.Bijector` instance, return `tfb.Chain([self, input])`.
3. Otherwise, return `self.forward(input)`

Args	
`value`	A `tfd.Distribution`, `tfb.Bijector`, or a (structure of) `Tensor`.
`name`	Python `str` name given to ops created by this function.
`**kwargs`	Additional keyword arguments passed into the created `tfd.TransformedDistribution`, `tfb.Bijector`, or `self.forward`.

Returns	
`composition`	A `tfd.TransformedDistribution` if the input was a `tfd.Distribution`, a `tfb.Chain` if the input was a `tfb.Bijector`, or a (structure of) `Tensor` computed by `self.forward`.

Examples

```
sigmoid = tfb.Reciprocal()(
    tfb.Shift(shift=1.)(
      tfb.Exp()(
        tfb.Scale(scale=-1.))))
# ==> `tfb.Chain([
#         tfb.Reciprocal(),
#         tfb.Shift(shift=1.),
#         tfb.Exp(),
#         tfb.Scale(scale=-1.),
#      ])`  # ie, `tfb.Sigmoid()`

log_normal = tfb.Exp()(tfd.Normal(0, 1))
# ==> `tfd.TransformedDistribution(tfd.Normal(0, 1), tfb.Exp())`

tfb.Exp()([-1., 0., 1.])
# ==> tf.exp([-1., 0., 1.])
```

`__eq__`

View source
(https://github.com/tensorflow/probability/blob/master/tensorflow_probability/python/bijectors/bijector.py#L827-L862)

```
__eq__(
    other
)
```

Return self==value.

Last updated 2021-08-26 UTC.