tf.contrib.distributions.bijectors.SoftmaxCentered

Contents

Class SoftmaxCentered
Properties
dtype
event_ndims

Class `SoftmaxCentered`

Inherits From: `Bijector`

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

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

Bijector which computes `Y = g(X) = exp([X 0]) / sum(exp([X 0]))`.

To implement softmax as a bijection, the forward transformation appends a value to the input and the inverse removes this coordinate. The appended coordinate represents a pivot, e.g., `softmax(x) = exp(x-c) / sum(exp(x-c))` where `c` is the implicit last coordinate.

Because we append a coordinate, this bijector only supports `event_ndim in [0, 1]`, i.e., scalars and vectors.

Example Use:

```
bijector.SoftmaxCentered(event_ndims=1).forward(tf.log([2, 3, 4]))
# Result: [0.2, 0.3, 0.4, 0.1]
# Extra result: 0.1

bijector.SoftmaxCentered(event_ndims=1).inverse([0.2, 0.3, 0.4, 0.1])
# Result: tf.log([2, 3, 4])
# Extra coordinate removed.
```

At first blush it may seem like the Invariance of domain theorem implies this implementation is not a bijection. However, the appended dimension makes the (forward) image non-open and the theorem does not directly apply.

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=0,
    validate_args=False,
    name='softmax_centered'
)
```

`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.