tf.contrib.distributions.TransformedDistribution

Contents

Class `TransformedDistribution`

Inherits From: `Distribution`

Defined in `tensorflow/python/ops/distributions/transformed_distribution.py`.

See the guide: Statistical Distributions (contrib) > Transformed distributions

A Transformed Distribution.

A `TransformedDistribution` models `p(y)` given a base distribution `p(x)`, and a deterministic, invertible, differentiable transform, `Y = g(X)`. The transform is typically an instance of the `Bijector` class and the base distribution is typically an instance of the `Distribution` class.

A `Bijector` is expected to implement the following functions: - `forward`, - `inverse`, - `inverse_log_det_jacobian`. The semantics of these functions are outlined in the `Bijector` documentation.

We now describe how a `TransformedDistribution` alters the input/outputs of a `Distribution` associated with a random variable (rv) `X`.

Write `cdf(Y=y)` for an absolutely continuous cumulative distribution function of random variable `Y`; write the probability density function `pdf(Y=y) := d^k / (dy_1,...,dy_k) cdf(Y=y)` for its derivative wrt to `Y` evaluated at `y`. Assume that `Y = g(X)` where `g` is a deterministic diffeomorphism, i.e., a non-random, continuous, differentiable, and invertible function. Write the inverse of `g` as `X = g^{-1}(Y)` and `(J o g)(x)` for the Jacobian of `g` evaluated at `x`.

A `TransformedDistribution` implements the following operations:

- `sample` Mathematically: `Y = g(X)` Programmatically: `bijector.forward(distribution.sample(...))`
- `log_prob` Mathematically: `(log o pdf)(Y=y) = (log o pdf o g^{-1})(y) + (log o abs o det o J o g^{-1})(y)` Programmatically: `(distribution.log_prob(bijector.inverse(y)) + bijector.inverse_log_det_jacobian(y))`
- `log_cdf` Mathematically: `(log o cdf)(Y=y) = (log o cdf o g^{-1})(y)` Programmatically: `distribution.log_cdf(bijector.inverse(x))`
- and similarly for: `cdf`, `prob`, `log_survival_function`, `survival_function`.

A simple example constructing a Log-Normal distribution from a Normal distribution:

```
ds = tf.contrib.distributions
log_normal = ds.TransformedDistribution(
  distribution=ds.Normal(loc=0., scale=1.),
  bijector=ds.bijectors.Exp(),
  name="LogNormalTransformedDistribution")
```

A `LogNormal` made from callables:

```
ds = tf.contrib.distributions
log_normal = ds.TransformedDistribution(
  distribution=ds.Normal(loc=0., scale=1.),
  bijector=ds.bijectors.Inline(
    forward_fn=tf.exp,
    inverse_fn=tf.log,
    inverse_log_det_jacobian_fn=(
      lambda y: -tf.reduce_sum(tf.log(y), axis=-1)),
  name="LogNormalTransformedDistribution")
```

Another example constructing a Normal from a StandardNormal:

```
ds = tf.contrib.distributions
normal = ds.TransformedDistribution(
  distribution=ds.Normal(loc=0., scale=1.),
  bijector=ds.bijectors.Affine(
    shift=-1.,
    scale_identity_multiplier=2.,
    event_ndims=0),
  name="NormalTransformedDistribution")
```

A `TransformedDistribution`'s batch- and event-shape are implied by the base distribution unless explicitly overridden by `batch_shape` or `event_shape` arguments. Specifying an overriding `batch_shape` (`event_shape`) is permitted only if the base distribution has scalar batch-shape (event-shape). The bijector is applied to the distribution as if the distribution possessed the overridden shape(s). The following example demonstrates how to construct a multivariate Normal as a `TransformedDistribution`.

```
ds = tf.contrib.distributions
# We will create two MVNs with batch_shape = event_shape = 2.
mean = [[-1., 0],      # batch:0
        [0., 1]]       # batch:1
chol_cov = [[[1., 0],
             [0, 1]],  # batch:0
            [[1, 0],
             [2, 2]]]  # batch:1
mvn1 = ds.TransformedDistribution(
    distribution=ds.Normal(loc=0., scale=1.),
    bijector=ds.bijectors.Affine(shift=mean, scale_tril=chol_cov),
    batch_shape=[2],  # Valid because base_distribution.batch_shape == [].
    event_shape=[2])  # Valid because base_distribution.event_shape == [].
mvn2 = ds.MultivariateNormalTriL(loc=mean, scale_tril=chol_cov)
# mvn1.log_prob(x) == mvn2.log_prob(x)
```

Properties

`allow_nan_stats`

Python `bool` describing behavior when a stat is undefined.

Stats return +/- infinity when it makes sense. E.g., the variance of a Cauchy distribution is infinity. However, sometimes the statistic is undefined, e.g., if a distribution's pdf does not achieve a maximum within the support of the distribution, the mode is undefined. If the mean is undefined, then by definition the variance is undefined. E.g. the mean for Student's T for df = 1 is undefined (no clear way to say it is either + or - infinity), so the variance = E[(X - mean)**2] is also undefined.

Returns:

- `allow_nan_stats` : Python `bool` .

`batch_shape`

Shape of a single sample from a single event index as a `TensorShape` .

May be partially defined or unknown.

The batch dimensions are indexes into independent, non-identical parameterizations of this distribution.

Returns:

- `batch_shape` : `TensorShape` , possibly unknown.

`bijector`

Function transforming x => y.

`distribution`

Base distribution, p(x).

`dtype`

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

`event_shape`

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

May be partially defined or unknown.

Returns:

- `event_shape` : `TensorShape` , possibly unknown.

`name`

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

`parameters`

Dictionary of parameters used to instantiate this `Distribution` .

`reparameterization_type`

Describes how samples from the distribution are reparameterized.

Currently this is one of the static instances `distributions.FULLY_REPARAMETERIZED` or `distributions.NOT_REPARAMETERIZED`.

Returns:

An instance of `ReparameterizationType`.

`validate_args`

Python `bool` indicating possibly expensive checks are enabled.

Methods

`__init__`

```
__init__(
    distribution,
    bijector=None,
    batch_shape=None,
    event_shape=None,
    validate_args=False,
    name=None
)
```

Construct a Transformed Distribution.

Args:

- `distribution`: The base distribution instance to transform. Typically an instance of `Distribution`.
- `bijector`: The object responsible for calculating the transformation. Typically an instance of `Bijector`. `None` means `Identity()`.
- `batch_shape`: `integer` vector `Tensor` which overrides `distribution` `batch_shape`; valid only if `distribution.is_scalar_batch()`.
- `event_shape`: `integer` vector `Tensor` which overrides `distribution` `event_shape`; valid only if `distribution.is_scalar_event()`.
- `validate_args`: Python `bool`, default `False`. When `True` distribution parameters are checked for validity despite possibly degrading runtime performance. When `False` invalid inputs may silently render incorrect outputs.
- `name`: Python `str` name prefixed to Ops created by this class. Default: `bijector.name + distribution.name`.

`batch_shape_tensor`

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

Shape of a single sample from a single event index as a 1-D `Tensor`.

The batch dimensions are indexes into independent, non-identical parameterizations of this distribution.

Args:

- `name`: name to give to the op

Returns:

- `batch_shape`: `Tensor`.

cdf

```
cdf(
    value,
    name='cdf'
)
```

Cumulative distribution function.

Given random variable `X`, the cumulative distribution function `cdf` is:

```
cdf(x) := P[X <= x]
```

Args:

- `value`: `float` or `double` `Tensor`.
- `name`: The name to give this op.

Returns:

- `cdf`: a `Tensor` of shape `sample_shape(x) + self.batch_shape` with values of type `self.dtype`.

copy

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

Creates a deep copy of the distribution.

Note: the copy distribution 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:

- `distribution`: 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)`.

covariance

```
covariance(name='covariance')
```

Covariance.

Covariance is (possibly) defined only for non-scalar-event distributions.

For example, for a length-`k`, vector-valued distribution, it is calculated as,

```
Cov[i, j] = Covariance(X_i, X_j) = E[(X_i - E[X_i]) (X_j - E[X_j])]
```

where `Cov` is a (batch of) `k x k` matrix, `0 <= (i, j) < k`, and `E` denotes expectation.

Alternatively, for non-vector, multivariate distributions (e.g., matrix-valued, Wishart), `Covariance` shall return a (batch of) matrices under some vectorization of the events, i.e.,

```
Cov[i, j] = Covariance(Vec(X)_i, Vec(X)_j) = [as above]
```

where `Cov` is a (batch of) `k' x k'` matrices, `0 <= (i, j) < k' = reduce_prod(event_shape)`, and `Vec` is some function mapping indices of this distribution's event dimensions to indices of a length-`k'` vector.

Args:

- `name`: The name to give this op.

Returns:

- `covariance`: Floating-point `Tensor` with shape `[B1, ..., Bn, k', k']` where the first `n` dimensions are batch coordinates and `k' = reduce_prod(self.event_shape)`.

`entropy`

```
entropy(name='entropy')
```

Shannon entropy in nats.

`event_shape_tensor`

```
event_shape_tensor(name='event_shape_tensor')
```

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

Args:

- `name`: name to give to the op

Returns:

- `event_shape`: `Tensor`.

`is_scalar_batch`

```
is_scalar_batch(name='is_scalar_batch')
```

Indicates that `batch_shape == []`.

Args:

- `name`: The name to give this op.

Returns:

* `is_scalar_batch`: `bool` scalar `Tensor`.

`is_scalar_event`

```
is_scalar_event(name='is_scalar_event')
```

Indicates that `event_shape == []`.

Args:

* `name`: The name to give this op.

Returns:

* `is_scalar_event`: `bool` scalar `Tensor`.

`log_cdf`

```
log_cdf(
    value,
    name='log_cdf'
)
```

Log cumulative distribution function.

Given random variable `X`, the cumulative distribution function `cdf` is:

```
log_cdf(x) := Log[ P[X <= x] ]
```

Often, a numerical approximation can be used for `log_cdf(x)` that yields a more accurate answer than simply taking the logarithm of the `cdf` when `x << -1`.

Args:

* `value`: `float` or `double` `Tensor`.
* `name`: The name to give this op.

Returns:

* `logcdf`: a `Tensor` of shape `sample_shape(x) + self.batch_shape` with values of type `self.dtype`.

`log_prob`

```
log_prob(
    value,
    name='log_prob'
)
```

Log probability density/mass function.

Args:

- `value`: `float` or `double` `Tensor`.
- `name`: The name to give this op.

Returns:

- `log_prob`: a `Tensor` of shape `sample_shape(x) + self.batch_shape` with values of type `self.dtype`.

`log_survival_function`

```
log_survival_function(
    value,
    name='log_survival_function'
)
```

Log survival function.

Given random variable `X`, the survival function is defined:

```
log_survival_function(x) = Log[ P[X > x] ]
                         = Log[ 1 - P[X <= x] ]
                         = Log[ 1 - cdf(x) ]
```

Typically, different numerical approximations can be used for the log survival function, which are more accurate than `1 - cdf(x)` when `x >> 1`.

Args:

- `value`: `float` or `double` `Tensor`.
- `name`: The name to give this op.

Returns:

`Tensor` of shape `sample_shape(x) + self.batch_shape` with values of type `self.dtype`.

`mean`

```
mean(name='mean')
```

Mean.

`mode`

```
mode(name='mode')
```

Mode.

`param_shapes`

```
param_shapes(
    cls,
    sample_shape,
    name='DistributionParamShapes'
)
```

Shapes of parameters given the desired shape of a call to `sample()`.

This is a class method that describes what key/value arguments are required to instantiate the given `Distribution` so that a particular shape is returned for that instance's call to `sample()`.

Subclasses should override class method `_param_shapes`.

Args:

- `sample_shape`: `Tensor` or python list/tuple. Desired shape of a call to `sample()`.
- `name`: name to prepend ops with.

Returns:

`dict` of parameter name to `Tensor` shapes.

`param_static_shapes`

```
param_static_shapes(
    cls,
    sample_shape
)
```

param_shapes with static (i.e. `TensorShape`) shapes.

This is a class method that describes what key/value arguments are required to instantiate the given `Distribution` so that a particular shape is returned for that instance's call to `sample()`. Assumes that the sample's shape is known statically.

Subclasses should override class method `_param_shapes` to return constant-valued tensors when constant values are fed.

Args:

- `sample_shape`: `TensorShape` or python list/tuple. Desired shape of a call to `sample()`.

Returns:

`dict` of parameter name to `TensorShape`.

Raises:

- `ValueError`: if `sample_shape` is a `TensorShape` and is not fully defined.

`prob`

```
prob(
    value,
    name='prob'
)
```

Probability density/mass function.

Args:

- `value`: `float` or `double` `Tensor`.
- `name`: The name to give this op.

Returns:

- `prob`: a `Tensor` of shape `sample_shape(x) + self.batch_shape` with values of type `self.dtype`.

`quantile`

```
quantile(
    value,
    name='quantile'
)
```

Quantile function. Aka "inverse cdf" or "percent point function".

Given random variable `X` and `p in [0, 1]`, the `quantile` is:

```
quantile(p) := x such that P[X <= x] == p
```

Args:

- `value`: `float` or `double` `Tensor`.
- `name`: The name to give this op.

Returns:

- `quantile`: a `Tensor` of shape `sample_shape(x) + self.batch_shape` with values of type `self.dtype`.

`sample`

```
sample(
    sample_shape=(),
    seed=None,
    name='sample'
)
```

Generate samples of the specified shape.

Note that a call to `sample()` without arguments will generate a single sample.

Args:

- `sample_shape`: 0D or 1D `int32` `Tensor`. Shape of the generated samples.

- `seed`: Python integer seed for RNG
- `name`: name to give to the op.

Returns:

- `samples`: a `Tensor` with prepended dimensions `sample_shape`.

`stddev`

```
stddev(name='stddev')
```

Standard deviation.

Standard deviation is defined as,

```
stddev = E[(X - E[X])**2]**0.5
```

where `X` is the random variable associated with this distribution, `E` denotes expectation, and `stddev.shape = batch_shape + event_shape`.

Args:

- `name`: The name to give this op.

Returns:

- `stddev`: Floating-point `Tensor` with shape identical to `batch_shape + event_shape`, i.e., the same shape as `self.mean()`.

`survival_function`

```
survival_function(
    value,
    name='survival_function'
)
```

Survival function.

Given random variable `X`, the survival function is defined:

```
survival_function(x) = P[X > x]
                     = 1 - P[X <= x]
                     = 1 - cdf(x).
```

Args:

- `value`: `float` or `double` `Tensor`.
- `name`: The name to give this op.

Returns:

`Tensor` of shape `sample_shape(x) + self.batch_shape` with values of type `self.dtype`.

`variance`

```
variance(name='variance')
```

Variance.

Variance is defined as,

```
Var = E[(X - E[X])**2]
```

where `X` is the random variable associated with this distribution, `E` denotes expectation, and **`Var.shape = batch_shape + event_shape`**.

Args:

- `name`: The name to give this op.

Returns:

- `variance`: Floating-point **`Tensor`** with shape identical to **`batch_shape + event_shape`**, i.e., the same shape as **`self.mean()`**.
