tf.contrib.distributions.SinhArcsinh

Contents

Class **SinhArcsinh**

Inherits From: `TransformedDistribution`

Defined in `tensorflow/contrib/distributions/python/ops/sinh_arcsinh.py`.

The SinhArcsinh transformation of a distribution on `(-inf, inf)`.

This distribution models a random variable, making use of a `SinhArcsinh` transformation (which has adjustable tailweight and skew), a rescaling, and a shift.

The `SinhArcsinh` transformation of the Normal is described in great depth in Sinh-arcsinh distributions. Here we use a slightly different parameterization, in terms of `tailweight` and `skewness`. Additionally we allow for distributions other than Normal, and control over `scale` as well as a "shift" parameter `loc`.

Mathematical Details

Given random variable `Z`, we define the SinhArcsinh transformation of `Z`, `Y`, parameterized by `(loc, scale, skewness, tailweight)`, via the relation:

```
Y := loc + scale * F(Z) * (2 / F(2))
F(Z) := Sinh( (Arcsinh(Z) + skewness) * tailweight )
```

This distribution is similar to the location-scale transformation `L(Z) := loc + scale * Z` in the following ways:

- If `skewness = 0` and `tailweight = 1` (the defaults), `F(Z) = Z`, and then `Y = L(Z)` exactly.
- `loc` is used in both to shift the result by a constant factor.
- Our definition of `C` ensures that `P[Y - loc <= 2 * scale] = P[L(Z) - loc <= 2 * scale]`. Thus it can be said that the weights in the tails of `Y` and `L(Z)` beyond `loc + 2 * scale` are the same.

This distribution is different than `loc + scale * Z` due to the reshaping done by `F`:

- Positive (negative) `skewness` leads to positive (negative) skew.
- positive skew means, the mode of `F(Z)` is "tilted" to the right.
- positive skew means positive values of `F(Z)` become more likely, and negative values become less likely.
- Larger (smaller) `tailweight` leads to fatter (thinner) tails.
- Fatter tails mean larger values of `|F(Z)|` become more likely.
- `tailweight < 1` leads to a distribution that is "flat" around `Y = loc`, and a very steep drop-off in the tails.

- `tailweight > 1` leads to a distribution more peaked at the mode with heavier tails.

To see the argument about the tails, note that for `|Z| >> 1` and `|Z| >> (|skewness| * tailweight)**tailweight`, we have `Y approx 0.5 Z**tailweight e**(sign(Z) skewness * tailweight)`.

To see the argument about `C` and quantiles, note that

```
P[(Y - loc) / scale <= 2] = P[F(Z) <= 2 * scale / C]
                           = P[Z <= F^{-1}(2 * scale / C)]
                           = P[Z <= 2].
```

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.

`loc`

The `loc` in `Y := loc + scale @ F(Z) * (2 / F(2)).

`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`.

`scale`

The `LinearOperator` `scale` in `Y := loc + scale @ F(Z) * (2 / F(2)).

`skewness`

Controls the skewness. `Skewness > 0` means right skew.

`tailweight`

Controls the tail decay. `tailweight > 1` means faster than Normal.

`validate_args`

Python `bool` indicating possibly expensive checks are enabled.

Methods

`__init__`

```
__init__(
    loc,
    scale,
    skewness=None,
    tailweight=None,
    distribution=None,
    validate_args=False,
    allow_nan_stats=True,
    name='MultivariateNormalLinearOperator'
)
```

Construct SinhArcsinh distribution on `(-inf, inf)`.

Arguments `(loc, scale, skewness, tailweight)` must have broadcastable shape (indexing batch dimensions). They must all have the same `dtype`.

Args:

- `loc`: Floating-point `Tensor`.
- `scale`: `Tensor` of same `dtype` as `loc`.
- `skewness`: Skewness parameter. Default is `0.0` (no skew).
- `tailweight`: Tailweight parameter. Default is `1.0` (unchanged tailweight)
- `distribution`: `tf.Distribution`-like instance. Distribution that is transformed to produce this distribution. Default is `ds.Normal(0., 1.)`. Must be a scalar-batch, scalar-event distribution. Typically `distribution.reparameterization_type = FULLY_REPARAMETERIZED` or it is a function of non-trainable parameters. WARNING: If you backprop through a `SinhArcsinh` sample and `distribution` is not `FULLY_REPARAMETERIZED` yet is a function of trainable variables, then the gradient will be incorrect!
- `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.
- `allow_nan_stats`: Python `bool`, default `True`. When `True`, statistics (e.g., mean, mode, variance) use the value "`NaN`" to indicate the result is undefined. When `False`, an exception is raised if one or more of the statistic's batch members are undefined.
- `name`: Python `str` name prefixed to Ops created by this class.

`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()`.

Last updated November 2, 2017.