tf.distributions.Distribution

Contents

Class `Distribution`

Aliases:

- Class `tf.contrib.distributions.Distribution`
- Class `tf.distributions.Distribution`

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

See the guide: Statistical Distributions (contrib) > Base classes

A generic probability distribution base class.

`Distribution` is a base class for constructing and organizing properties (e.g., mean, variance) of random variables (e.g, Bernoulli, Gaussian).

Subclassing

Subclasses are expected to implement a leading-underscore version of the same-named function. The argument signature should be identical except for the omission of `name="..."`. For example, to enable `log_prob(value, name="log_prob")` a subclass should implement `_log_prob(value)`.

Subclasses can append to public-level docstrings by providing docstrings for their method specializations. For example:

```
@util.AppendDocstring("Some other details.")
def _log_prob(self, value):
  ...
```

would add the string "Some other details." to the `log_prob` function docstring. This is implemented as a simple decorator to avoid python linter complaining about missing Args/Returns/Raises sections in the partial docstrings.

Broadcasting, batching, and shapes

All distributions support batches of independent distributions of that type. The batch shape is determined by broadcasting together the parameters.

The shape of arguments to `__init__`, `cdf`, `log_cdf`, `prob`, and `log_prob` reflect this broadcasting, as does the return value of `sample` and `sample_n`.

`sample_n_shape = [n] + batch_shape + event_shape`, where `sample_n_shape` is the shape of the `Tensor` returned from `sample_n`, `n` is the number of samples, `batch_shape` defines how many independent distributions there are, and `event_shape` defines the shape of samples from each of those independent distributions. Samples are independent along the `batch_shape` dimensions, but not necessarily so along the `event_shape` dimensions (depending on the particulars of the underlying distribution).

Using the `Uniform` distribution as an example:

```
minval = 3.0
maxval = [[4.0, 6.0],
          [10.0, 12.0]]

# Broadcasting:
# This instance represents 4 Uniform distributions. Each has a lower bound at
# 3.0 as the `minval` parameter was broadcasted to match `maxval`'s shape.
u = Uniform(minval, maxval)

# `event_shape` is `TensorShape([])`.
event_shape = u.event_shape
# `event_shape_t` is a `Tensor` which will evaluate to [].
event_shape_t = u.event_shape_tensor()

# Sampling returns a sample per distribution. `samples` has shape
# [5, 2, 2], which is [n] + batch_shape + event_shape, where n=5,
# batch_shape=[2, 2], and event_shape=[].
samples = u.sample_n(5)

# The broadcasting holds across methods. Here we use `cdf` as an example. The
# same holds for `log_cdf` and the likelihood functions.

# `cum_prob` has shape [2, 2] as the `value` argument was broadcasted to the
# shape of the `Uniform` instance.
cum_prob_broadcast = u.cdf(4.0)

# `cum_prob`'s shape is [2, 2], one per distribution. No broadcasting
# occurred.
cum_prob_per_dist = u.cdf([[4.0, 5.0],
                           [6.0, 7.0]])

# INVALID as the `value` argument is not broadcastable to the distribution's
# shape.
cum_prob_invalid = u.cdf([4.0, 5.0, 6.0])
```

Parameter values leading to undefined statistics or distributions.

Some distributions do not have well-defined statistics for all initialization parameter values. For example, the beta distribution is parameterized by positive real numbers `concentration1` and `concentration0`, and does not have well-defined mode if `concentration1 < 1` or `concentration0 < 1`.

The user is given the option of raising an exception or returning `NaN`.

```
a = tf.exp(tf.matmul(logits, weights_a))
b = tf.exp(tf.matmul(logits, weights_b))

# Will raise exception if ANY batch member has a < 1 or b < 1.
dist = distributions.beta(a, b, allow_nan_stats=False)
mode = dist.mode().eval()

# Will return NaN for batch members with either a < 1 or b < 1.
dist = distributions.beta(a, b, allow_nan_stats=True)  # Default behavior
mode = dist.mode().eval()
```

In all cases, an exception is raised if *invalid* parameters are passed, e.g.

```
# Will raise an exception if any Op is run.
negative_a = -1.0 * a  # beta distribution by definition has a > 0.
dist = distributions.beta(negative_a, b, allow_nan_stats=True)
dist.mean().eval()
```

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.

`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__(
    dtype,
    reparameterization_type,
    validate_args,
    allow_nan_stats,
    parameters=None,
    graph_parents=None,
    name=None
)
```

Constructs the `Distribution`.

This is a private method for subclass use.

Args:

- `dtype`: The type of the event samples. `None` implies no type-enforcement.
- `reparameterization_type`: Instance of `ReparameterizationType`. If `distributions.FULLY_REPARAMETERIZED`, this `Distribution` can be reparameterized in terms of some standard distribution with a function whose Jacobian is constant for the support of the standard distribution. If `distributions.NOT_REPARAMETERIZED`, then no such reparameterization is available.
- `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.
- `parameters`: Python `dict` of parameters used to instantiate this `Distribution`.
- `graph_parents`: Python `list` of graph prerequisites of this `Distribution`.
- `name`: Python `str` name prefixed to Ops created by this class. Default: subclass name.

Raises:

- `ValueError`: if any member of graph_parents is **`None`** or not a **`Tensor`**.

`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`

```
@classmethod
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`

```
@classmethod
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()`.