tf.distributions.Beta

Contents

Class `Beta`

Inherits From: `Distribution`

Aliases:

- Class `tf.contrib.distributions.Beta`
- Class `tf.distributions.Beta`

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

See the guide: Statistical Distributions (contrib) > Univariate (scalar) distributions

Beta distribution.

The Beta distribution is defined over the `(0, 1)` interval using parameters `concentration1` (aka "alpha") and `concentration0` (aka "beta").

Mathematical Details

The probability density function (pdf) is,

```
pdf(x; alpha, beta) = x**(alpha - 1) (1 - x)**(beta - 1) / Z
Z = Gamma(alpha) Gamma(beta) / Gamma(alpha + beta)
```

where:

- `concentration1 = alpha`,
- `concentration0 = beta`,
- `Z` is the normalization constant, and,
- `Gamma` is the gamma function.

The concentration parameters represent mean total counts of a `1` or a `0`, i.e.,

```
concentration1 = alpha = mean * total_concentration
concentration0 = beta  = (1. - mean) * total_concentration
```

where `mean` in `(0, 1)` and `total_concentration` is a positive real number representing a mean `total_count = concentration1 + concentration0`.

Distribution parameters are automatically broadcast in all functions; see examples for details.

Examples

```
# Create a batch of three Beta distributions.
alpha = [1, 2, 3]
beta = [1, 2, 3]
dist = Beta(alpha, beta)

dist.sample([4, 5])  # Shape [4, 5, 3]

# `x` has three batch entries, each with two samples.
x = [[.1, .4, .5],
     [.2, .3, .5]]
# Calculate the probability of each pair of samples under the corresponding
# distribution in `dist`.
dist.prob(x)         # Shape [2, 3]
```

```
# Create batch_shape=[2, 3] via parameter broadcast:
alpha = [[1.], [2]]      # Shape [2, 1]
beta = [3., 4, 5]        # Shape [3]
dist = Beta(alpha, beta)

# alpha broadcast as: [[1., 1, 1,],
#                      [2, 2, 2]]
# beta broadcast as:  [[3., 4, 5],
#                      [3, 4, 5]]
# batch_Shape [2, 3]
dist.sample([4, 5])  # Shape [4, 5, 2, 3]

x = [.2, .3, .5]
# x will be broadcast as [[.2, .3, .5],
#                         [.2, .3, .5]],
# thus matching batch_shape [2, 3].
dist.prob(x)         # Shape [2, 3]
```

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.

`concentration0`

Concentration parameter associated with a `0` outcome.

`concentration1`

Concentration parameter associated with a `1` outcome.

`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`.

`total_concentration`

Sum of concentration parameters.

`validate_args`

Python `bool` indicating possibly expensive checks are enabled.

Methods

`__init__`

```
__init__(
    concentration1=None,
    concentration0=None,
    validate_args=False,
    allow_nan_stats=True,
    name='Beta'
)
```

Initialize a batch of Beta distributions.

Args:

- `concentration1` : Positive floating-point `Tensor` indicating mean number of successes; aka "alpha". Implies `self.dtype` and `self.batch_shape`, i.e., `concentration1.shape = [N1, N2, ..., Nm] = self.batch_shape`.
- `concentration0` : Positive floating-point `Tensor` indicating mean number of failures; aka "beta". Otherwise has same semantics as `concentration1`.
- `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]
```

Additional documentation from `Beta`:

> **Note:** `x` must have dtype `self.dtype` and be in `[0, 1].` It must have a shape compatible with `self.batch_shape()`.

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`.

Additional documentation from `Beta`:

> ★ **Note:** x must have dtype `self.dtype` and be in `[0, 1]`. It must have a shape compatible with `self.batch_shape()`.

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.

Additional documentation from `Beta`:

> **Note:** x must have dtype `self.dtype` and be in `[0, 1]`. It must have a shape compatible with `self.batch_shape()`.

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.

Additional documentation from `Beta`:

> **Note:** The mode is undefined when `concentration1 <= 1` or `concentration0 <= 1`. If `self.allow_nan_stats` is `True`, `NaN` is used for undefined modes. If `self.allow_nan_stats` is `False` an exception is raised when one or more modes are undefined.

`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.

Additional documentation from `Beta`:

Note: x must have dtype `self.dtype` and be in `[0, 1]`. It must have a shape compatible with `self.batch_shape()`.

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()`.

Except as otherwise noted, the content of this page is licensed under the Creative Commons Attribution 3.0 License, and code samples are licensed under the Apache 2.0 License. For details, see our Site Policies. Java is a registered trademark of Oracle and/or its affiliates.

Last updated November 2, 2017.