tf.contrib.distributions.QuantizedDistribution

Contents

Class QuantizedDistribution
Properties
allow_nan_stats
batch_shape

Class `QuantizedDistribution`

Inherits From: `Distribution`

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

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

Distribution representing the quantization `Y = ceiling(X)`.

Definition in terms of sampling.

```
1. Draw X
2. Set Y <-- ceiling(X)
3. If Y < low, reset Y <-- low
4. If Y > high, reset Y <-- high
5. Return Y
```

Definition in terms of the probability mass function.

Given scalar random variable `X`, we define a discrete random variable `Y` supported on the integers as follows:

```
P[Y = j] := P[X <= low],  if j == low,
         := P[X > high - 1],  j == high,
         := 0, if j < low or j > high,
         := P[j - 1 < X <= j],  all other j.
```

Conceptually, without cutoffs, the quantization process partitions the real line `R` into half open intervals, and identifies an integer `j` with the right endpoints:

```
R = ... (-2, -1](-1, 0](0, 1](1, 2](2, 3](3, 4] ...
j = ...      -1      0     1     2     3     4  ...
```

`P[Y = j]` is the mass of `X` within the `jth` interval. If `low = 0`, and `high = 2`, then the intervals are redrawn and `j` is re-assigned:

```
R = (-infty, 0](0, 1](1, infty)
j =          0     1     2
```

`P[Y = j]` is still the mass of `X` within the `jth` interval.

Caveats

Since evaluation of each `P[Y = j]` involves a cdf evaluation (rather than a closed form function such as for a Poisson), computations such as mean and entropy are better done with samples or approximations, and are not implemented by this class.

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.

`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,
    low=None,
    high=None,
    validate_args=False,
    name='QuantizedDistribution'
)
```

Construct a Quantized Distribution representing `Y = ceiling(X)`.

Some properties are inherited from the distribution defining `X`. Example: `allow_nan_stats` is determined for this `QuantizedDistribution` by reading the `distribution`.

Args:

- `distribution`: The base distribution class to transform. Typically an instance of `Distribution`.
- `low`: `Tensor` with same `dtype` as this distribution and shape able to be added to samples. Should be a whole number. Default `None`. If provided, base distribution's `prob` should be defined at `low`.
- `high`: `Tensor` with same `dtype` as this distribution and shape able to be added to samples. Should be a whole number. Default `None`. If provided, base distribution's `prob` should be defined at `high - 1`. `high` must be strictly greater than `low`.
- `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.

Raises:

- `TypeError` : If `dist_cls` is not a subclass of `Distribution` or continuous.
- `NotImplementedError` : If the base distribution does not implement `cdf` .

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 `QuantizedDistribution` :

For whole numbers `y` ,

```
cdf(y) := P[Y <= y]
        = 1, if y >= high,
        = 0, if y < low,
        = P[X <= y], otherwise.
```

Since `Y` only has mass at whole numbers, `P[Y <= y] = P[Y <= floor(y)]` . This dictates that fractional `y` are first floored to a whole number, and then above definition applies.

The base distribution's `cdf` method must be defined on `y - 1` .

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 `QuantizedDistribution`:

For whole numbers `y`,

```
cdf(y) := P[Y <= y]
        = 1, if y >= high,
        = 0, if y < low,
        = P[X <= y], otherwise.
```

Since `Y` only has mass at whole numbers, `P[Y <= y] = P[Y <= floor(y)]`. This dictates that fractional `y` are first floored to a whole number, and then above definition applies.

The base distribution's `log_cdf` method must be defined on `y - 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.

Additional documentation from `QuantizedDistribution`:

For whole numbers `y`,

```
P[Y = y] := P[X <= low],  if y == low,
         := P[X > high - 1],  y == high,
         := 0, if j < low or y > high,
         := P[y - 1 < X <= y],  all other y.
```

The base distribution's `log_cdf` method must be defined on `y - 1`. If the base distribution has a `log_survival_function` method results will be more accurate for large values of `y`, and in this case the `log_survival_function` must also be defined on `y - 1`.

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`.

Additional documentation from `QuantizedDistribution`:

For whole numbers `y`,

```
survival_function(y) := P[Y > y]
                      = 0, if y >= high,
                      = 1, if y < low,
                      = P[X <= y], otherwise.
```

Since `Y` only has mass at whole numbers, `P[Y <= y] = P[Y <= floor(y)]`. This dictates that fractional `y` are first floored to a whole number, and then above definition applies.

The base distribution's `log_cdf` method must be defined on `y - 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.

Additional documentation from `QuantizedDistribution`:

For whole numbers `y`,

```
P[Y = y] := P[X <= low],  if y == low,
         := P[X > high - 1],  y == high,
         := 0, if j < low or y > high,
         := P[y - 1 < X <= y],  all other y.
```

The base distribution's `cdf` method must be defined on `y - 1`. If the base distribution has a `survival_function` method, results will be more accurate for large values of `y`, and in this case the `survival_function` must also be defined on `y - 1`.

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).
```

Additional documentation from `QuantizedDistribution`:

For whole numbers `y`,

```
survival_function(y) := P[Y > y]
                      = 0, if y >= high,
                      = 1, if y < low,
                      = P[X <= y], otherwise.
```

Since `Y` only has mass at whole numbers, `P[Y <= y] = P[Y <= floor(y)]`. This dictates that fractional `y` are first floored to a whole number, and then above definition applies.

The base distribution's `cdf` method must be defined on `y - 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`.

`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.