tf.contrib.distributions.PoissonLogNormalQuadratureCompound

Contents

Class **PoissonLogNormalQuadratureCompound**

Inherits From: `Distribution`

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

`PoissonLogNormalQuadratureCompound` distribution.

The `PoissonLogNormalQuadratureCompound` is an approximation to a Poisson-LogNormal compound distribution, i.e.,

```
p(k|loc, scale)
= int_{R_+} dl LogNormal(l | loc, scale) Poisson(k | l)
= int_{R} dz ((lambda(z) sqrt(2) scale)
              * exp(-z**2) / (lambda(z) sqrt(2 pi) sigma)
              * Poisson(k | lambda(z)))
= int_{R} dz exp(-z**2) / sqrt(pi) Poisson(k | lambda(z))
approx= sum{ prob[d] Poisson(k | lambda(grid[d])) : d=0, ..., deg-1 }
```

where `lambda(z) = exp(sqrt(2) scale z + loc)` and the `prob,grid` terms are from Gauss–Hermite quadrature. Note that the second line made the substitution: `z(l) = (log(l) - loc) / (sqrt(2) scale)` which implies `lambda(z)` [above] and `dl = sqrt(2) scale lambda(z) dz`

In the non-approximation case, a draw from the LogNormal prior represents the Poisson rate parameter. Unfortunately, the non-approximate distribution lacks an analytical probability density function (pdf). Therefore the `PoissonLogNormalQuadratureCompound` class implements an approximation based on Gauss-Hermite quadrature. Note: although the `PoissonLogNormalQuadratureCompound` is approximately the Poisson-LogNormal compound distribution, it is itself a valid distribution. Viz., it possesses a `sample`, `log_prob`, `mean`, `variance`, etc. which are all mutually consistent.

Mathematical Details

The `PoissonLogNormalQuadratureCompound` approximates a Poisson-LogNormal compound distribution. Using variable-substitution and Gauss-Hermite quadrature we can redefine the distribution to be a parameter-less convex combination of `deg` different Poisson samples.

That is, defined over positive integers, this distribution is parameterized by a (batch of) `loc` and `scale` scalars.

The probability density function (pdf) is,

```
pdf(k | loc, scale, deg)
  = sum{ prob[d] Poisson(k | lambda=exp(sqrt(2) scale grid[d] + loc))
        : d=0, ..., deg-1 }
```

where, `grid, w = numpy.polynomial.hermite.hermgauss(deg)` and `prob = w / sqrt(pi)`.

Examples

```
ds = tf.contrib.distributions
# Create two batches of PoissonLogNormalQuadratureCompounds, one with
# prior `loc = 0.` and another with `loc = 1.` In both cases `scale = 1.`
pln = ds.PoissonLogNormalQuadratureCompound(
    loc=[0., -0.5],
    scale=1.,
    quadrature_polynomial_degree=10,
    validate_args=True)

## Properties

<h3 id="allow_nan_stats"><code>allow_nan_stats</code></h3>

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:

* <b>`allow_nan_stats`</b>: Python `bool`.

<h3 id="batch_shape"><code>batch_shape</code></h3>

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:

* <b>`batch_shape`</b>: `TensorShape`, possibly unknown.

<h3 id="distribution"><code>distribution</code></h3>

Base Poisson parameterized by a Gauss-Hermite grid of rates.

<h3 id="dtype"><code>dtype</code></h3>

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

<h3 id="event_shape"><code>event_shape</code></h3>

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

May be partially defined or unknown.

#### Returns:

* <b>`event_shape`</b>: `TensorShape`, possibly unknown.

<h3 id="loc"><code>loc</code></h3>
```

````
Location parameter of the LogNormal prior.

<h3 id="mixture_distribution"><code>mixture_distribution</code></h3>

Distribution which randomly selects a Poisson with Gauss-Hermite rate.

<h3 id="name"><code>name</code></h3>

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

<h3 id="parameters"><code>parameters</code></h3>

Dictionary of parameters used to instantiate this `Distribution`.

<h3 id="quadrature_polynomial_degree"><code>quadrature_polynomial_degree</code></h3>

Polynomial largest exponent used for Gauss-Hermite quadrature.

<h3 id="reparameterization_type"><code>reparameterization_type</code></h3>

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`.

<h3 id="scale"><code>scale</code></h3>

Scale parameter of the LogNormal prior.

<h3 id="validate_args"><code>validate_args</code></h3>

Python `bool` indicating possibly expensive checks are enabled.

## Methods

<h3 id="__init__"><code>__init__</code></h3>

``` python
__init__(
    loc,
    scale,
    quadrature_polynomial_degree=8,
    validate_args=False,
    allow_nan_stats=True,
    name='PoissonLogNormalQuadratureCompound'
)
````

Constructs the PoissonLogNormalQuadratureCompound on `R**k`.

Args:

- `loc`: `float`-like (batch of) scalar `Tensor`; the location parameter of the LogNormal prior.
- `scale`: `float`-like (batch of) scalar `Tensor`; the scale parameter of the LogNormal prior.
- `quadrature_polynomial_degree`: Python `int`-like scalar. Default value: 8.
- `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.

Raises:

- `TypeError`: if `loc.dtype != scale[0].dtype`.

`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.