tfp.distributions.MixtureSameFamily

Mixture (same-family) distribution.

Inherits From: **`Distribution`**
(https://www.tensorflow.org/probability/api_docs/python/tfp/distributions/Distribution)

```
tfp.distributions.MixtureSameFamily(
    mixture_distribution, components_distribution, reparameterize=False,
    validate_args=False, allow_nan_stats=True, name='MixtureSameFamily'
)
```

The **`MixtureSameFamily`** distribution implements a (batch of) mixture distribution where all components are from different parameterizations of the same distribution type. It is parameterized by a **`Categorical`** 'selecting distribution' (over **`k`** components) and a components distribution, i.e., a **`Distribution`** with a rightmost batch shape (equal to **`[k]`**) which indexes each (batch of) component.

Examples

```
tfd = tfp.distributions

### Create a mixture of two scalar Gaussians:

gm = tfd.MixtureSameFamily(
    mixture_distribution=tfd.Categorical(
        probs=[0.3, 0.7]),
    components_distribution=tfd.Normal(
      loc=[-1., 1],       # One for each component.
      scale=[0.1, 0.5]))  # And same here.
```

```
gm.mean()
# ==> 0.4

gm.variance()
# ==> 1.018

# Plot PDF.
x = np.linspace(-2., 3., int(1e4), dtype=np.float32)
import matplotlib.pyplot as plt
plt.plot(x, gm.prob(x));

### Create a mixture of two Bivariate Gaussians:

gm = tfd.MixtureSameFamily(
    mixture_distribution=tfd.Categorical(
        probs=[0.3, 0.7]),
    components_distribution=tfd.MultivariateNormalDiag(
        loc=[[-1., 1],  # component 1
             [1, -1]],  # component 2
        scale_identity_multiplier=[.3, .6]))

gm.mean()
# ==> array([ 0.4, -0.4], dtype=float32)

gm.covariance()
# ==> array([[ 1.119, -0.84],
#            [-0.84,  1.119]], dtype=float32)

# Plot PDF contours.
def meshgrid(x, y=x):
  [gx, gy] = np.meshgrid(x, y, indexing='ij')
  gx, gy = np.float32(gx), np.float32(gy)
  grid = np.concatenate([gx.ravel()[None, :], gy.ravel()[None, :]], axis=0)
  return grid.T.reshape(x.size, y.size, 2)
grid = meshgrid(np.linspace(-2, 2, 100, dtype=np.float32))
plt.contour(grid[..., 0], grid[..., 1], gm.prob(grid));
```

Args

`mixture_distribution`	[tfp.distributions.Categorical](https://www.tensorflow.org/probability/api_docs/python/tfp/distributions/Categorical)-like instance. Manages the probability of selecting components. The number of categories must match the rightmost batch dimension of the `components_distribution`. Must have either scalar `batch_`

	`shape` or `batch_shape` matching `components_distribution.batch_shape[:-1]`.
`components_distribution`	`tfp.distributions.Distribution` (https://www.tensorflow.org/probability/api_docs/python/tfp/distributions/Distribution) -like instance. Right-most batch dimension indexes components.
`reparameterize`	Python `bool`, default `False`. Whether to reparameterize samples of the distribution using implicit reparameterization gradients [(Figurnov et al., 2018)][1]. The gradients for the mixture logits are equivalent to the ones described by [(Graves, 2016)][2]. The gradients for the components parameters are also computed using implicit reparameterization (as opposed to ancestral sampling), meaning that all components are updated every step. Only works when: (1) components_distribution is fully reparameterized; (2) components_distribution is either a scalar distribution or fully factorized (tfd.Independent applied to a scalar distribution); (3) batch shape has a known rank. Experimental, may be slow and produce infs/NaNs.
`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

`ValueError`	`if not dtype_util.is_integer(mixture_distribution.dtype)`.
`ValueError`	if mixture_distribution does not have scalar `event_shape`.
`ValueError`	if `mixture_distribution.batch_shape` and `components_distribution.batch_shape[:-1]` are both fully defined and the former is neither scalar nor equal to the latter.
`ValueError`	if `mixture_distribution` categories does not equal `components_distribution` rightmost batch shape.

Attributes

`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.
`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.
`components_distribution`	
`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.
`experimental_is_sharded`	
`experimental_shard_axis_names`	The list or structure of lists of active shard axis names.
`mixture_distribution`	
`name`	Name prepended to all ops created by this `Distribution`.
`name_scope`	Returns a `tf.name_scope` (https://www.tensorflow.org/api_docs/python/tf/name_scope) instance for this class.
`non_trainable_variables`	Sequence of non-trainable variables owned by this module and its submodules. **Note:** this method uses reflection to find variables on the current instance and submodules. For performance reasons you may wish to cache the result of calling this method if you don't expect the return value to change.
`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 `tfd.FULLY_REPARAMETERIZED` or `tfd.NOT_REPARAMETERIZED`.

`submodules`	Sequence of all sub-modules.
	
	Submodules are modules which are properties of this module, or found as properties of modules which are properties of this module (and so on).
	
	`>>> a = tf.Module()`
	`>>> b = tf.Module()`
	`>>> c = tf.Module()`
	`>>> a.b = b`
	`>>> b.c = c`
	`>>> list(a.submodules) == [b, c]`
	`True`
	`>>> list(b.submodules) == [c]`
	`True`
	`>>> list(c.submodules) == []`
	`True`

`trainable_variables`	Sequence of trainable variables owned by this module and its submodules. **Note:** this method uses reflection to find variables on the current instance and submodules. For performance reasons you may wish to cache the result of calling this method if you don't expect the return value to change.
`validate_args`	Python `bool` indicating possibly expensive checks are enabled.
`variables`	Sequence of variables owned by this module and its submodules. **Note:** this method uses reflection to find variables on the current instance and submodules. For performance reasons you may wish to cache the result of calling this method if you don't expect the return value to change.

Attributes

`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.
`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.
`components_distribution`	
`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.
`experimental_is_sharded`	
`experimental_shard_axis_names`	The list or structure of lists of active shard axis names.
`mixture_distribution`	
`name`	Name prepended to all ops created by this `Distribution`.
`name_scope`	Returns a `tf.name_scope` (https://www.tensorflow.org/api_docs/python/tf/name_scope) instance for this class.
`non_trainable_variables`	Sequence of non-trainable variables owned by this module and its submodules. **Note:** this method uses reflection to find variables on the current instance and submodules. For performance reasons you may wish to cache the result of calling this method if you don't expect the return value to change.
`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 `tfd.FULLY_REPARAMETERIZED` or `tfd.NOT_REPARAMETERIZED`.

`submodules`	Sequence of all sub-modules.
	
	Submodules are modules which are properties of this module, or found as properties of modules which are properties of this module (and so on).
	
	`>>> a = tf.Module()`
	`>>> b = tf.Module()`
	`>>> c = tf.Module()`
	`>>> a.b = b`
	`>>> b.c = c`
	`>>> list(a.submodules) == [b, c]`
	`True`
	`>>> list(b.submodules) == [c]`
	`True`
	`>>> list(c.submodules) == []`
	`True`

`trainable_variables`	Sequence of trainable variables owned by this module and its submodules. **Note:** this method uses reflection to find variables on the current instance and submodules. For performance reasons you may wish to cache the result of calling this method if you don't expect the return value to change.
`validate_args`	Python `bool` indicating possibly expensive checks are enabled.
`variables`	Sequence of variables owned by this module and its submodules. **Note:** this method uses reflection to find variables on the current instance and submodules. For performance reasons you may wish to cache the result of calling this method if you don't expect the return value to change.

Methods

`batch_shape_tensor`

View source
(https://github.com/tensorflow/probability/blob/v0.13.0/tensorflow_probability/python/distributions/distribution.py#L1026-L1064)

```
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

View source
(https://github.com/tensorflow/probability/blob/v0.13.0/tensorflow_probability/python/distributions/distribution.py#L1420-L1438)

```
cdf(
    value, name='cdf', **kwargs
)
```

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`	Python `str` prepended to names of ops created by this function.
`**kwargs`	Named arguments forwarded to subclass implementation.

Returns

`cdf`	a `Tensor` of shape `sample_shape(x) + self.batch_shape` with values of type `self.dtype`.

copy

View source (https://github.com/tensorflow/probability/blob/v0.13.0/tensorflow_probability/python/distributions/distribution.py#L921-L950)

```
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

View source (https://github.com/tensorflow/probability/blob/v0.13.0/tensorflow_probability/python/distributions/distribution.py#L1657-L1695)

```
covariance(
    name='covariance', **kwargs
)
```

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`	Python `str` prepended to names of ops created by this function.
`**kwargs`	Named arguments forwarded to subclass implementation.

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)`.

cross_entropy

View source (https://github.com/tensorflow/probability/blob/v0.13.0/tensorflow_probability/python/distributions/distribution.py#L1710-L1733)

```
cross_entropy(
    other, name='cross_entropy'
)
```

Computes the (Shannon) cross entropy.

Denote this distribution (`self`) by `P` and the `other` distribution by `Q`. Assuming `P, Q` are absolutely continuous with respect to one another and permit densities `p(x) dr(x)` and `q(x) dr(x)`, (Shannon) cross entropy is defined as:

```
H[P, Q] = E_p[-log q(X)] = -int_F p(x) log q(x) dr(x)
```

where `F` denotes the support of the random variable `X ~ P`.

Args	
`other`	`tfp.distributions.Distribution` (https://www.tensorflow.org/probability/api_docs/python/tfp/distributions/Distribution) instance.
`name`	Python `str` prepended to names of ops created by this function.

Returns	
`cross_entropy`	`self.dtype` `Tensor` with shape `[B1, ..., Bn]` representing `n` different calculations of (Shannon) cross entropy.

entropy

View source (https://github.com/tensorflow/probability/blob/v0.13.0/tensorflow_probability/python/distributions/distribution.py#L1533-L1536)

```
entropy(
    name='entropy', **kwargs
)
```

Shannon entropy in nats.

event_shape_tensor

View source (https://github.com/tensorflow/probability/blob/v0.13.0/tensorflow_probability/python/distributions/distribution.py#L1161-L1187)

```
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`.

experimental_default_event_space_bijector

View source (https://github.com/tensorflow/probability/blob/v0.13.0/tensorflow_probability/python/distributions/distribution.py#L1783-L1812)

```
experimental_default_event_space_bijector(
    *args, **kwargs
)
```

Bijector mapping the reals (R**n) to the event space of the distribution.

Distributions with continuous support may implement `_default_event_space_bijector` which returns a subclass of `tfp.bijectors.Bijector` (https://www.tensorflow.org/probability/api_docs/python/tfp/bijectors/AutoCompositeTensorBijector) that maps R**n to the distribution's event space. For example, the default bijector for the `Beta` distribution is `tfp.bijectors.Sigmoid()` (https://www.tensorflow.org/probability/api_docs/python/tfp/bijectors/Sigmoid), which maps the real line to `[0, 1]`, the support of the `Beta` distribution. The default bijector for the `CholeskyLKJ` distribution is `tfp.bijectors.CorrelationCholesky` (https://www.tensorflow.org/probability/api_docs/python/tfp/bijectors/CorrelationCholesky), which maps R^(k * (k-1) // 2) to the submanifold of k x k lower triangular matrices with ones along the diagonal.

The purpose of `experimental_default_event_space_bijector` is to enable gradient descent in an unconstrained space for Variational Inference and Hamiltonian Monte Carlo methods. Some effort has been made to choose bijectors such that the tails of the distribution in the unconstrained space are between Gaussian and Exponential.

For distributions with discrete event space, or for which TFP currently lacks a suitable bijector, this function returns `None`.

Args	
`*args`	Passed to implementation `_default_event_space_bijector`.
`**kwargs`	Passed to implementation `_default_event_space_bijector`.

Returns	
`event_space_bijector`	`Bijector` instance or `None`.

is_scalar_batch

View source (https://github.com/tensorflow/probability/blob/v0.13.0/tensorflow_probability/python/distributions/distribution.py#L1218-L1230)

```
is_scalar_batch(
    name='is_scalar_batch'
)
```

Indicates that `batch_shape == []`.

Args

`name`	Python `str` prepended to names of ops created by this function.

Returns

`is_scalar_batch`	`bool` scalar `Tensor`.

is_scalar_event

View source (https://github.com/tensorflow/probability/blob/v0.13.0/tensorflow_probability/python/distributions/distribution.py#L1204-L1216)

```
is_scalar_event(
    name='is_scalar_event'
)
```

Indicates that `event_shape == []`.

Args

`name`	Python `str` prepended to names of ops created by this function.

Returns

`is_scalar_event`	`bool` scalar `Tensor`.

kl_divergence

View source (https://github.com/tensorflow/probability/blob/v0.13.0/tensorflow_probability/python/distributions/distribution.py#L1739-L1770)

```
kl_divergence(
    other, name='kl_divergence'
)
```

Computes the Kullback--Leibler divergence.

Denote this distribution (`self`) by `p` and the `other` distribution by `q`. Assuming `p, q` are absolutely continuous with respect to reference measure `r`, the KL divergence is defined as:

```
KL[p, q] = E_p[log(p(X)/q(X))]
         = -int_F p(x) log q(x) dr(x) + int_F p(x) log p(x) dr(x)
         = H[p, q] - H[p]
```

where `F` denotes the support of the random variable `X ~ p`, `H[., .]` denotes (Shannon) cross entropy, and `H[.]` denotes (Shannon) entropy.

Args	
`other`	tfp.distributions.Distribution (https://www.tensorflow.org/probability/api_docs/python/tfp/distributions/Distribution) instance.
`name`	Python `str` prepended to names of ops created by this function.

Returns	
`kl_divergence`	`self.dtype` `Tensor` with shape `[B1, ..., Bn]` representing `n` different calculations of the Kullback-Leibler divergence.

log_cdf

View source (https://github.com/tensorflow/probability/blob/v0.13.0/tensorflow_probability/python/distributions/distribution.py#L1382-L1404)

```
log_cdf(
    value, name='log_cdf', **kwargs
)
```

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`	Python `str` prepended to names of ops created by this function.
`**kwargs`	Named arguments forwarded to subclass implementation.

Returns

`logcdf`	a `Tensor` of shape `sample_shape(x) + self.batch_shape` with values of type `self.dtype`.

log_prob

View source (https://github.com/tensorflow/probability/blob/v0.13.0/tensorflow_probability/python/distributions/distribution.py#L1284-L1296)

```
log_prob(
    value, name='log_prob', **kwargs
)
```

Log probability density/mass function.

Args

`value`	`float` or `double` `Tensor`.
`name`	Python `str` prepended to names of ops created by this function.
`**kwargs`	Named arguments forwarded to subclass implementation.

Returns

`log_prob`	a `Tensor` of shape `sample_shape(x) + self.batch_shape` with values of type `self.dtype`.

log_survival_function

View source (https://github.com/tensorflow/probability/blob/v0.13.0/tensorflow_probability/python/distributions/distribution.py#L1461-L1485)

```
log_survival_function(
    value, name='log_survival_function', **kwargs
)
```

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`	Python `str` prepended to names of ops created by this function.
`**kwargs`	Named arguments forwarded to subclass implementation.

Returns

`Tensor` of shape `sample_shape(x) + self.batch_shape` with values of type `self.dtype`.

mean

View source
(https://github.com/tensorflow/probability/blob/v0.13.0/tensorflow_probability/python/distributions/distribution.py#L1542-L1545)

```
mean(
    name='mean', **kwargs
)
```

Mean.

mode

View source
(https://github.com/tensorflow/probability/blob/v0.13.0/tensorflow_probability/python/distributions/distribution.py#L1701-L1704)

```
mode(
    name='mode', **kwargs
)
```

Mode.

param_shapes

View source
(https://github.com/tensorflow/probability/blob/v0.13.0/tensorflow_probability/python/distributions/distribution.py#L730-L756)

```
@classmethod
param_shapes(
    sample_shape, name='DistributionParamShapes'
)
```

Shapes of parameters given the desired shape of a call to `sample()`. (deprecated)

ıg: THIS FUNCTION IS DEPRECATED. It will be removed after 2021-03-01. Instructions for updating: The **`_shapes`** method of **`tfd.Distribution`** is deprecated; use **`parameter_properties`** instead.

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

View source (https://github.com/tensorflow/probability/blob/v0.13.0/tensorflow_probability/python/distributions/distribution.py#L758-L798)

```
@classmethod
param_static_shapes(
    sample_shape
)
```

param_shapes with static (i.e. `TensorShape`) shapes. (deprecated)

ıg: THIS FUNCTION IS DEPRECATED. It will be removed after 2021-03-01. Instructions for updating: The **`_static_shapes`** method of **`tfd.Distribution`** is deprecated; use **`parameter_properties`** instead.

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.

`parameter_properties`

View source (https://github.com/tensorflow/probability/blob/v0.13.0/tensorflow_probability/python/distributions/distribution.py#L684-L728)

```
@classmethod
parameter_properties(
    dtype=tf.float32, num_classes=None
)
```

Returns a dict mapping constructor arg names to property annotations.

This dict should include an entry for each of the distribution's `Tensor`-valued constructor arguments.

Distribution subclasses are not required to implement `_parameter_properties`, so this method may raise `NotImplementedError`. Providing a `_parameter_properties` implementation enables several advanced features, including:

- Distribution batch slicing (`sliced_distribution = distribution[i:j]`).
- Automatic inference of `_batch_shape` and `_batch_shape_tensor`, which must otherwise be computed explicitly.
- Automatic instantiation of the distribution within TFP's internal property tests.
- Automatic construction of 'trainable' instances of the distribution using appropriate bijectors to avoid violating parameter constraints. This enables the distribution family to be used easily as a surrogate posterior in variational inference.

In the future, parameter property annotations may enable additional functionality; for example, returning Distribution instances from `tf.vectorized_map` (https://www.tensorflow.org/api_docs/python/tf/vectorized_map).

Args

`dtype`	Optional float `dtype` to assume for continuous-valued parameters. Some constraining bijectors require advance knowledge of the dtype because certain constants (e.g., `tfb.Softplus.low`) must be instantiated with the same dtype as the values to be transformed.
`num_classes`	Optional `int` `Tensor` number of classes to assume when inferring the shape of parameters for categorical-like distributions. Otherwise ignored.

Returns

`parameter_properties`	A `str -` >tfp.python.internal.parameter_properties.ParameterProperties`dict mapping constructor argument names to`ParameterProperties` instances.

Raises

`NotImplementedError`	if the distribution class does not implement `_parameter_properties`.

prob

View source (https://github.com/tensorflow/probability/blob/v0.13.0/tensorflow_probability/python/distributions/distribution.py#L1312-L1324)

```
prob(
    value, name='prob', **kwargs
)
```

Probability density/mass function.

Args

`value`	`float` or `double` `Tensor`.
`name`	Python `str` prepended to names of ops created by this function.
`**kwargs`	Named arguments forwarded to subclass implementation.

Returns

`prob`	a `Tensor` of shape `sample_shape(x) + self.batch_shape` with values of type `self.dtype`.

quantile

View source (https://github.com/tensorflow/probability/blob/v0.13.0/tensorflow_probability/python/distributions/distribution.py#L1564-L1582)

```
quantile(
    value, name='quantile', **kwargs
)
```

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`	Python `str` prepended to names of ops created by this function.
`**kwargs`	Named arguments forwarded to subclass implementation.

Returns

`quantile`	a `Tensor` of shape `sample_shape(x) + self.batch_shape` with values of type `self.dtype`.

sample

View source (https://github.com/tensorflow/probability/blob/v0.13.0/tensorflow_probability/python/distributions/distribution.py#L1253-L1268)

```
sample(
    sample_shape=(), seed=None, name='sample', **kwargs
)
```

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 or `tfp.util.SeedStream` (https://www.tensorflow.org/probability/api_docs/python/tfp/util/SeedStream) instance, for seeding PRNG.
`name`	name to give to the op.
`**kwargs`	Named arguments forwarded to subclass implementation.

Returns

`samples`	a `Tensor` with prepended dimensions `sample_shape`.

stddev

View source (https://github.com/tensorflow/probability/blob/v0.13.0/tensorflow_probability/python/distributions/distribution.py#L1622-L1651)

```
stddev(
    name='stddev', **kwargs
)
```

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`	Python `str` prepended to names of ops created by this function.
`**kwargs`	Named arguments forwarded to subclass implementation.

Returns

`stddev`	Floating-point `Tensor` with shape identical to `batch_shape + event_shape`, i.e., the same shape as `self.mean()`.

survival_function

View source (https://github.com/tensorflow/probability/blob/v0.13.0/tensorflow_probability/python/distributions/distribution.py#L1507-L1527)

```
survival_function(
    value, name='survival_function', **kwargs
)
```

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`	Python `str` prepended to names of ops created by this function.
`**kwargs`	Named arguments forwarded to subclass implementation.

Returns
`Tensor` of shape `sample_shape(x) + self.batch_shape` with values of type `self.dtype`.

unnormalized_log_prob

View source (https://github.com/tensorflow/probability/blob/v0.13.0/tensorflow_probability/python/distributions/distribution.py#L1344-L1366)

```
unnormalized_log_prob(
    value, name='unnormalized_log_prob', **kwargs
)
```

Potentially unnormalized log probability density/mass function.

This function is similar to `log_prob`, but does not require that the return value be normalized. (Normalization here refers to the total integral of probability being one, as it should be by definition for any probability distribution.) This is useful, for example, for distributions where the normalization constant is difficult or expensive to compute. By default, this simply calls `log_prob`.

Args	
`value`	`float` or `double` `Tensor`.
`name`	Python `str` prepended to names of ops created by this function.
`**kwargs`	Named arguments forwarded to subclass implementation.

Returns	
`unnormalized_log_prob`	a `Tensor` of shape `sample_shape(x) + self.batch_shape` with values of type `self.dtype`.

variance

View source (https://github.com/tensorflow/probability/blob/v0.13.0/tensorflow_probability/python/distributions/distribution.py#L1588-L1616)

```
variance(
    name='variance', **kwargs
)
```

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`	Python `str` prepended to names of ops created by this function.
`**kwargs`	Named arguments forwarded to subclass implementation.

Returns	
`variance`	Floating-point `Tensor` with shape identical to `batch_shape + event_shape`, i.e., the same shape as `self.mean()`.

with_name_scope

```
@classmethod
with_name_scope(
    method
)
```

Decorator to automatically enter the module name scope.

```
>>> class MyModule(tf.Module):
...   @tf.Module.with_name_scope
...   def __call__(self, x):
...     if not hasattr(self, 'w'):
...       self.w = tf.Variable(tf.random.normal([x.shape[1], 3]))
...     return tf.matmul(x, self.w)
```

Using the above module would produce `tf.Variable` (https://www.tensorflow.org/api_docs/python/tf/Variable)s and `tf.Tensor` (https://www.tensorflow.org/api_docs/python/tf/Tensor)s whose names included the module name:

```
>>> mod = MyModule()
>>> mod(tf.ones([1, 2]))
<tf.Tensor: shape=(1, 3), dtype=float32, numpy=..., dtype=float32)>
>>> mod.w
<tf.Variable 'my_module/Variable:0' shape=(2, 3) dtype=float32,
numpy=..., dtype=float32)>
```

Args

`method`	The method to wrap.

Returns

The original method wrapped such that it enters the module's name scope.

`__getitem__`

View source (https://github.com/tensorflow/probability/blob/v0.13.0/tensorflow_probability/python/distributions/mixture_same_family.py#L228-L253)

```
__getitem__(
    slices
)
```

Slices the batch axes of this distribution, returning a new instance.

```
b = tfd.Bernoulli(logits=tf.zeros([3, 5, 7, 9]))
b.batch_shape  # => [3, 5, 7, 9]
b2 = b[:, tf.newaxis, ..., -2:, 1::2]
b2.batch_shape  # => [3, 1, 5, 2, 4]

x = tf.random.normal([5, 3, 2, 2])
cov = tf.matmul(x, x, transpose_b=True)
chol = tf.linalg.cholesky(cov)
loc = tf.random.normal([4, 1, 3, 1])
mvn = tfd.MultivariateNormalTriL(loc, chol)
mvn.batch_shape  # => [4, 5, 3]
mvn.event_shape  # => [2]
mvn2 = mvn[:, 3:, ..., ::-1, tf.newaxis]
mvn2.batch_shape  # => [4, 2, 3, 1]
mvn2.event_shape  # => [2]
```

Args	
`slices`	slices from the [] operator

Returns	
`dist`	A new `tfd.Distribution` instance with sliced parameters.

`__iter__`

View source (https://github.com/tensorflow/probability/blob/v0.13.0/tensorflow_probability/python/distributions/distribution.py#L879-L880)

```
__iter__()
```