

## tf.distributions.Normal

## Contents

Class Normal

Aliases:

Properties

allow\_nan\_stats

Class **Normal**Inherits From: [Distribution](#)

## Aliases:

- Class [tf.contrib.distributions.Normal](#)
- Class [tf.distributions.Normal](#)

Defined in [tensorflow/python/ops/distributions/normal.py](#).See the guide: [Statistical Distributions \(contrib\) > Univariate \(scalar\) distributions](#)The Normal distribution with location **loc** and **scale** parameters.

## Mathematical details

The probability density function (pdf) is,

```
pdf(x; mu, sigma) = exp(-0.5 (x - mu)**2 / sigma**2) / Z
Z = (2 pi sigma**2)**0.5
```

where **loc** = **mu** is the mean, **scale** = **sigma** is the std. deviation, and, **Z** is the normalization constant.The Normal distribution is a member of the [location-scale family](#), i.e., it can be constructed as,

```
X ~ Normal(loc=0, scale=1)
Y = loc + scale * X
```

## Examples

Examples of initialization of one or a batch of distributions.

```
# Define a single scalar Normal distribution.
dist = tf.distributions.Normal(loc=0., scale=3.)

# Evaluate the cdf at 1, returning a scalar.
dist.cdf(1.)

# Define a batch of two scalar valued Normals.
# The first has mean 1 and standard deviation 11, the second 2 and 22.
dist = tf.distributions.Normal(loc=[1, 2.], scale=[11, 22.])

# Evaluate the pdf of the first distribution on 0, and the second on 1.5,
# returning a length two tensor.
dist.prob([0, 1.5])

# Get 3 samples, returning a 3 x 2 tensor.
dist.sample([3])
```

Arguments are broadcast when possible.

```
# Define a batch of two scalar valued Normals.
# Both have mean 1, but different standard deviations.
dist = tf.distributions.Normal(loc=1., scale=[11, 22.])

# Evaluate the pdf of both distributions on the same point, 3.0,
# returning a length 2 tensor.
dist.prob(3.0)
```

## 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 =  $E[(X - \text{mean})^2]$  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.

## **loc**

Distribution parameter for the mean.

## **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` .

## **scale**

Distribution parameter for standard deviation.

## **validate\_args**

Python `bool` indicating possibly expensive checks are enabled.

## Methods

---

### **`__init__`**

```
__init__(
    loc,
    scale,
    validate_args=False,
    allow_nan_stats=True,
    name='Normal'
)
```

Construct Normal distributions with mean and stddev `loc` and `scale` .

The parameters `loc` and `scale` must be shaped in a way that supports broadcasting (e.g. `loc + scale` is a valid operation).

Args:

- `loc` : Floating point tensor; the means of the distribution(s).
- `scale` : Floating point tensor; the stddevs of the distribution(s). Must contain only positive values.
- `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` and `scale` have different `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,  $0 \leq (i, j) < k$ , 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,  $0 \leq (i, j) < k' = \text{reduce\_prod}(\text{event\_shape})$ , 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:

$$\text{log\_cdf}(x) := \text{Log}[ P[X \leq x] ]$$

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  $1 - \text{cdf}(x)$  when  $x \gg 1$ .

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  $\mathbf{X}$  is the random variable associated with this distribution,  $\mathbf{E}$  denotes expectation, and  $\text{stddev.shape} = \text{batch\_shape} + \text{event\_shape}$ .

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  $\mathbf{X}$ , 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  $\mathbf{X}$  is the random variable associated with this distribution,  $\mathbf{E}$  denotes expectation, and  $\text{Var.shape} = \text{batch\_shape} + \text{event\_shape}$ .

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.

### Stay Connected

[Blog](#)

[GitHub](#)

[Twitter](#)

### Support

[Issue Tracker](#)

[Release Notes](#)

[Stack Overflow](#)

**English**

[Terms](#) | [Privacy](#)