

tf.contrib.distributions.MultivariateNormalDiagPlusLowRank

Contents

Class MultivariateNormalDiagPlusLowRank

Properties

allow_nan_stats

batch_shape

Class **MultivariateNormalDiagPlusLowRank**Defined in [tensorflow/contrib/distributions/python/ops/mvn_diag_plus_low_rank.py](#).See the guide: [Statistical Distributions \(contrib\)](#) > [Multivariate distributions](#)The multivariate normal distribution on \mathbb{R}^k .

The Multivariate Normal distribution is defined over \mathbb{R}^k and parameterized by a (batch of) length- k `loc` vector (aka "mu") and a (batch of) $k \times k$ `scale` matrix; `covariance = scale @ scale.T` where `@` denotes matrix-multiplication.

Mathematical Details

The probability density function (pdf) is,

```
pdf(x; loc, scale) = exp(-0.5 ||y||**2) / Z,
y = inv(scale) @ (x - loc),
Z = (2 pi)**(0.5 k) |det(scale)|,
```

where:

- `loc` is a vector in \mathbb{R}^k ,
- `scale` is a linear operator in $\mathbb{R}^{k \times k}$, `cov = scale @ scale.T`,
- `Z` denotes the normalization constant, and,
- `||y||**2` denotes the squared Euclidean norm of `y`.

A (non-batch) `scale` matrix is:

```
scale = diag(scale_diag + scale_identity_multiplier ones(k)) +
        scale_perturb_factor @ diag(scale_perturb_diag) @ scale_perturb_factor.T
```

where:

- `scale_diag.shape = [k]`,
- `scale_identity_multiplier.shape = []`,
- `scale_perturb_factor.shape = [k, r]`, typically $k \gg r$, and,
- `scale_perturb_diag.shape = [r]`.

Additional leading dimensions (if any) will index batches.

If both `scale_diag` and `scale_identity_multiplier` are `None`, then `scale` is the Identity matrix.

The MultivariateNormal distribution is a member of the [location-scale family](#), i.e., it can be constructed as,

```
X ~ MultivariateNormal(loc=0, scale=1)  # Identity scale, zero shift.
Y = scale @ X + loc
```

Examples

```
ds = tf.contrib.distributions

# Initialize a single 3-variate Gaussian with covariance `cov = S @ S.T`,
# `S = diag(d) + U @ diag(m) @ U.T`. The perturbation, `U @ diag(m) @ U.T`, is
# a rank-2 update.
mu = [-0.5., 0, 0.5]  # shape: [3]
d = [1.5, 0.5, 2]      # shape: [3]
U = [[1., 2],
      [-1, 1],
      [2, -0.5]]        # shape: [3, 2]
m = [4., 5]             # shape: [2]
mvn = ds.MultivariateNormalDiagPlusLowRank(
    loc=mu
    scale_diag=d
    scale_perturb_factor=U,
    scale_perturb_diag=m)

# Evaluate this on an observation in `R^3`, returning a scalar.
mvn.prob([-1, 0, 1]).eval()  # shape: []

# Initialize a 2-batch of 3-variate Gaussians; `S = diag(d) + U @ U.T`.
mu = [[1., 2, 3],
      [11, 22, 33]]        # shape: [b, k] = [2, 3]
U = [[[1., 2],
      [3, 4],
      [5, 6]],
      [[0.5, 0.75],
      [1, 0, 0.25],
      [1.5, 1.25]]]        # shape: [b, k, r] = [2, 3, 2]
m = [[0.1, 0.2],
      [0.4, 0.5]]          # shape: [b, r] = [2, 2]

mvn = ds.MultivariateNormalDiagPlusLowRank(
    loc=mu,
    scale_perturb_factor=U,
    scale_perturb_diag=m)

mvn.covariance().eval()  # shape: [2, 3, 3]
# ==> [[[ 15.63   31.57   48.51]
#        [ 31.57   69.31  105.05]
#        [ 48.51  105.05  162.59]]
#
#        [[ 2.59   1.41   3.35]
#         [ 1.41   2.71   3.34]
#         [ 3.35   3.34   8.35]]]

# Compute the pdf of two `R^3` observations (one from each batch);
# return a length-2 vector.
x = [[-0.9, 0, 0.1],
      [-10, 0, 9]]      # shape: [2, 3]
mvn.prob(x).eval()      # shape: [2]
```

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.

bijector

Function transforming $x \Rightarrow y$.

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.

loc

The `loc` `Tensor` in $Y = \text{scale} @ X + \text{loc}$.

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

The `scale` `LinearOperator` in $Y = \text{scale} @ X + \text{loc}$.

validate_args

Python `bool` indicating possibly expensive checks are enabled.

Methods

`__init__`

```
__init__(
    loc=None,
    scale_diag=None,
    scale_identity_multiplier=None,
    scale_perturb_factor=None,
    scale_perturb_diag=None,
    validate_args=False,
    allow_nan_stats=True,
    name='MultivariateNormalDiagPlusLowRank'
)
```

Construct Multivariate Normal distribution on \mathbf{R}^k .

The `batch_shape` is the broadcast shape between `loc` and `scale` arguments.

The `event_shape` is given by last dimension of the matrix implied by `scale`. The last dimension of `loc` (if provided) must broadcast with this.

Recall that `covariance = scale @ scale.T`. A (non-batch) `scale` matrix is:

```
scale = diag(scale_diag + scale_identity_multiplier ones(k)) +
        scale_perturb_factor @ diag(scale_perturb_diag) @ scale_perturb_factor.T
```

where:

- `scale_diag.shape = [k]`,
- `scale_identity_multiplier.shape = []`,
- `scale_perturb_factor.shape = [k, r]`, typically $k \gg r$, and,
- `scale_perturb_diag.shape = [r]`.

Additional leading dimensions (if any) will index batches.

If both `scale_diag` and `scale_identity_multiplier` are `None`, then `scale` is the Identity matrix.

Args:

- `loc`: Floating-point `Tensor`. If this is set to `None`, `loc` is implicitly `0`. When specified, may have shape `[B1, ..., Bb, k]` where $b \geq 0$ and k is the event size.
- `scale_diag`: Non-zero, floating-point `Tensor` representing a diagonal matrix added to `scale`. May have shape `[B1, ..., Bb, k]`, $b \geq 0$, and characterizes b -batches of $k \times k$ diagonal matrices added to `scale`. When both `scale_identity_multiplier` and `scale_diag` are `None` then `scale` is the `Identity`.
- `scale_identity_multiplier`: Non-zero, floating-point `Tensor` representing a scaled-identity-matrix added to `scale`. May have shape `[B1, ..., Bb]`, $b \geq 0$, and characterizes b -batches of scaled $k \times k$ identity matrices added to `scale`. When both `scale_identity_multiplier` and `scale_diag` are `None` then `scale` is the `Identity`.
- `scale_perturb_factor`: Floating-point `Tensor` representing a rank- r perturbation added to `scale`. May have shape `[B1, ..., Bb, k, r]`, $b \geq 0$, and characterizes b -batches of rank- r updates to `scale`. When `None`, no rank- r update is added to `scale`.
- `scale_perturb_diag`: Floating-point `Tensor` representing a diagonal matrix inside the rank- r perturbation added to `scale`. May have shape `[B1, ..., Bb, r]`, $b \geq 0$, and characterizes b -batches of $r \times r$ diagonal matrices inside the perturbation added to `scale`. When `None`, an identity matrix is used inside the perturbation. Can only be specified if `scale_perturb_factor` is also specified.
- `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 at most `scale_identity_multiplier` is specified.

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) $k \times k$ 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) $k' \times k'$ 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 $[B_1, \dots, B_n, k', k']$ where the first n dimensions are batch coordinates and $k' = \text{reduce_prod}(\text{self.event_shape})$.

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.

Additional documentation from `MultivariateNormalLinearOperator` :

`value` is a batch vector with compatible shape if `value` is a `Tensor` whose shape can be broadcast up to either:

```
self.batch_shape + self.event_shape
```

or

```
[M1, ..., Mm] + self.batch_shape + self.event_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.

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 `MultivariateNormalLinearOperator`:

`value` is a batch vector with compatible shape if `value` is a `Tensor` whose shape can be broadcast up to either:

```
self.batch_shape + self.event_shape
```

or

```
[M1, ..., Mm] + self.batch_shape + self.event_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,

$$\text{stddev} = E[(X - E[X])**2]**0.5$$

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.

Stay Connected

[Blog](#)

[GitHub](#)

[Twitter](#)

Support

[Issue Tracker](#)

[Release Notes](#)

[Stack Overflow](#)

English

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