

tf.contrib.distributions.ExpRelaxedOneHotCategorical

Contents

Class ExpRelaxedOneHotCategorical

Properties

allow_nan_stats

batch_shape

Class **ExpRelaxedOneHotCategorical**

Inherits From: [Distribution](#)Defined in [tensorflow/contrib/distributions/python/ops/relaxed_onehot_categorical.py](#).

ExpRelaxedOneHotCategorical distribution with temperature and logits.

An ExpRelaxedOneHotCategorical distribution is a log-transformed RelaxedOneHotCategorical distribution. The RelaxedOneHotCategorical is a distribution over random probability vectors, vectors of positive real values that sum to one, which continuously approximates a OneHotCategorical. The degree of approximation is controlled by a temperature: as the temperature goes to 0 the RelaxedOneHotCategorical becomes discrete with a distribution described by the logits, as the temperature goes to infinity the RelaxedOneHotCategorical becomes the constant distribution that is identically the constant vector of (1/event_size, ..., 1/event_size).

Because computing log-probabilities of the RelaxedOneHotCategorical can suffer from underflow issues, this class is one solution for loss functions that depend on log-probabilities, such as the KL Divergence found in the variational autoencoder loss. The KL divergence between two distributions is invariant under invertible transformations, so evaluating KL divergences of ExpRelaxedOneHotCategorical samples, which are always followed by a `tf.exp` op, is equivalent to evaluating KL divergences of RelaxedOneHotCategorical samples. See the appendix of Maddison et al., 2016 for more mathematical details, where this distribution is called the ExpConcrete.

Examples

Creates a continuous distribution, whose exp approximates a 3-class one-hot categorical distribution. The 2nd class is the most likely to be the largest component in samples drawn from this distribution. If those samples are followed by a `tf.exp` op, then they are distributed as a relaxed onehot categorical.

```
temperature = 0.5
p = [0.1, 0.5, 0.4]
dist = ExpRelaxedOneHotCategorical(temperature, probs=p)
samples = dist.sample()
exp_samples = tf.exp(samples)
# exp_samples has the same distribution as samples from
# RelaxedOneHotCategorical(temperature, probs=p)
```

Creates a continuous distribution, whose exp approximates a 3-class one-hot categorical distribution. The 2nd class is the most likely to be the largest component in samples drawn from this distribution.

```

temperature = 0.5
logits = [-2, 2, 0]
dist = ExpRelaxedOneHotCategorical(temperature, logits=logits)
samples = dist.sample()
exp_samples = tf.exp(samples)
# exp_samples has the same distribution as samples from
# RelaxedOneHotCategorical(temperature, probs=p)

```

Creates a continuous distribution, whose exp approximates a 3-class one-hot categorical distribution. Because the temperature is very low, samples from this distribution are almost discrete, with one component almost 0 and the others very negative. The 2nd class is the most likely to be the largest component in samples drawn from this distribution.

```

temperature = 1e-5
logits = [-2, 2, 0]
dist = ExpRelaxedOneHotCategorical(temperature, logits=logits)
samples = dist.sample()
exp_samples = tf.exp(samples)
# exp_samples has the same distribution as samples from
# RelaxedOneHotCategorical(temperature, probs=p)

```

Creates a continuous distribution, whose exp approximates a 3-class one-hot categorical distribution. Because the temperature is very high, samples from this distribution are usually close to the $(-\log(3), -\log(3), -\log(3))$ vector. The 2nd class is still the most likely to be the largest component in samples drawn from this distribution.

```

temperature = 10
logits = [-2, 2, 0]
dist = ExpRelaxedOneHotCategorical(temperature, logits=logits)
samples = dist.sample()
exp_samples = tf.exp(samples)
# exp_samples has the same distribution as samples from
# RelaxedOneHotCategorical(temperature, probs=p)

```

Chris J. Maddison, Andriy Mnih, and Yee Whye Teh. The Concrete Distribution: A Continuous Relaxation of Discrete Random Variables. 2016.

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.

event_size

Scalar `int32` tensor: the number of classes.

logits

Vector of coordinatewise logits.

name

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

parameters

Dictionary of parameters used to instantiate this `Distribution` .

probs

Vector of probabilities summing to one.

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` .

temperature

Batchwise temperature tensor of a RelaxedCategorical.

validate_args

Python `bool` indicating possibly expensive checks are enabled.

Methods

__init__

```
__init__(
    temperature,
    logits=None,
    probs=None,
    dtype=tf.float32,
    validate_args=False,
    allow_nan_stats=True,
    name='ExpRelaxedOneHotCategorical'
)
```

Initialize ExpRelaxedOneHotCategorical using class log-probabilities.

Args:

- `temperature`: An 0-D `Tensor`, representing the temperature of a set of ExpRelaxedCategorical distributions. The temperature should be positive.
- `logits`: An N-D `Tensor`, `N >= 1`, representing the log probabilities of a set of ExpRelaxedCategorical distributions. The first `N - 1` dimensions index into a batch of independent distributions and the last dimension represents a vector of logits for each class. Only one of `logits` or `probs` should be passed in.
- `probs`: An N-D `Tensor`, `N >= 1`, representing the probabilities of a set of ExpRelaxedCategorical distributions. The first `N - 1` dimensions index into a batch of independent distributions and the last dimension represents a vector of probabilities for each class. Only one of `logits` or `probs` should be passed in.
- `dtype`: The type of the event samples (default: float32).
- `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.

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.

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 $x \gg 1$.

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,

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

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:

$$\begin{aligned} \text{survival_function}(x) &= P[X > x] \\ &= 1 - P[X \leq x] \\ &= 1 - \text{cdf}(x). \end{aligned}$$

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,

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

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](#)