
Tars Documentation

Release 0.0.2

Author

Jul 05, 2018

1	Tars package	3
1.1	Subpackages	3
1.1.1	Tars.distributions package	3
1.1.1.1	Submodules	3
1.1.1.2	Tars.distributions.distribution_models module	3
1.1.1.3	Tars.distributions.distribution_samples module	5
1.1.1.4	Tars.distributions.estimate_kl module	10
1.1.1.5	Tars.distributions.multiple module	10
1.1.1.6	Module contents	12
1.1.2	Tars.layers package	16
1.1.2.1	Submodules	16
1.1.2.2	Tars.layers.conv_recurrent module	16
1.1.2.3	Tars.layers.recurrent module	16
1.1.2.4	Tars.layers.shape module	16
1.1.2.5	Module contents	16
1.1.3	Tars.models package	16
1.1.3.1	Submodules	16
1.1.3.2	Tars.models.ae module	16
1.1.3.3	Tars.models.gan module	16
1.1.3.4	Tars.models.model module	16
1.1.3.5	Tars.models.vae module	17
1.1.3.6	Module contents	17
1.1.4	Tars.tests package	18
1.1.4.1	Submodules	18
1.1.4.2	Tars.tests.test_utils module	18
1.1.4.3	Module contents	18
1.2	Submodules	18
1.3	Tars.load_data module	18
1.4	Tars.utils module	19
1.5	Module contents	19
2	Indices and tables	21
	Python Module Index	23

Contents:

1.1 Subpackages

1.1.1 Tars.distributions package

1.1.1.1 Submodules

1.1.1.2 Tars.distributions.distribution_models module

class Tars.distributions.distribution_models.**Bernoulli** (*mean_network*, *given*,
temp=0.1, *seed=1*)

Bases: *Tars.distributions.distribution_models.Distribution*

class Tars.distributions.distribution_models.**Beta** (*alpha_network*, *beta_network*,
given, *iter_sampling=6*, *rejection_sampling=True*, *seed=1*)

Bases: *Tars.distributions.distribution_models.DistributionDouble*

class Tars.distributions.distribution_models.**Categorical** (*mean_network*, *given*,
temp=0.1, *n_dim=1*,
seed=1)

Bases: *Tars.distributions.distribution_models.Distribution*

fprop (*x*, **args*, ***kwargs*)

x [list] This contains Theano variables, which must to correspond to ‘given’.

mean [Theano variable] The output of this distribution.

sample_given_x (*x*, *repeat=1*, ***kwargs*)

x [list] This contains Theano variables, which must to correspond to ‘given’.

repeat : int or theano variable

list This contains ‘x’ and sample $\sim p(*|x)$, such as [x, sample].

```
class Tars.distributions.distribution_models.Deterministic (network,          given,
                                                         seed=1)
    Bases: Tars.distributions.distribution_models.Distribution
```

```
class Tars.distributions.distribution_models.Dirichlet (alpha_network,      given,
                                                         k, iter_sampling=6, re-
                                                         jection_sampling=True,
                                                         seed=1)
    Bases: Tars.distributions.distribution_models.Distribution
```

```
sample_given_x (x, repeat=1, **kwargs)

    x [list] This contains Theano variables, which must to correspond to ‘given’.
    repeat : int or thenao variable
    list This contains ‘x’ and sample  $\sim p(\cdot|x)$ , such as [x, sample].
```

```
class Tars.distributions.distribution_models.Distribution (distribution,
                                                         mean_network,
                                                         given,          seed=1,
                                                         set_log_likelihood=True)

    Bases: object
```

```
mean_network [lasagne.layers.Layer] The network whose outputs express the paramater of this distribution.
given [list] This contains instances of lasagne.layers.InputLayer, which mean the conditioning variables. e.g. if
    given = [x,y], then the corresponding log-likelihood is
    log p( $\cdot|x,y$ )
fprop (x, *args, **kwargs)
    x [list] This contains Theano variables, which must to correspond to ‘given’.
    mean [Theano variable] The output of this distribution.

get_input_shape ()
    tuple This represents the shape of the inputs of this distribution.
get_output_shape ()
    tuple This represents the shape of the output of this distribution.
get_params ()
log_likelihood_given_x (samples, **kwargs)
    samples [list] This contains ‘x’, which has Theano variables, and test sample.
    Theano variable, shape (n_samples,) A log-likelihood, p(sample|x).

sample_given_x (x, repeat=1, **kwargs)
    x [list] This contains Theano variables, which must to correspond to ‘given’.
    repeat : int or thenao variable
    list This contains ‘x’ and sample  $\sim p(\cdot|x)$ , such as [x, sample].
sample_mean_given_x (x, *args, **kwargs)
    x [list] This contains Theano variables, which must to correspond to ‘given’.
    list This contains ‘x’ and a mean value of sample  $\sim p(\cdot|x)$ .
```



```

set_seed (seed=1)

class Tars.distributions.distribution_models.DistributionDouble (distribution,
                                                                mean_network,
                                                                var_network,
                                                                given,
                                                                seed=1)

Bases: Tars.distributions.distribution_models.Distribution

fprop (x, deterministic=False)

    x [list] This contains Theano variables, which must to correspond to ‘given’.

    mean [Theano variable] The output of this distribution.

get_params ()

class Tars.distributions.distribution_models.Gamma (alpha_network,    beta_network,
                                                    given, seed=1)
Bases: Tars.distributions.distribution_models.DistributionDouble

class Tars.distributions.distribution_models.Gaussian (mean_network, var_network,
                                                    given, seed=1)
Bases: Tars.distributions.distribution_models.DistributionDouble

class Tars.distributions.distribution_models.GaussianConstantVar (mean_network,
                                                                    given, var=1,
                                                                    seed=1)

Bases: Tars.distributions.distribution_models.Distribution

class Tars.distributions.distribution_models.Kumaraswamy (a_network,
                                                            b_network,    given,
                                                            stick_breaking=True,
                                                            seed=1)

Bases: Tars.distributions.distribution_models.DistributionDouble

[Naelisnick+ 2016] Deep Generative Models with Stick-Breaking Priors

log_likelihood_given_x (*args)

    samples [list] This contains ‘x’, which has Theano variables, and test sample.

    Theano variable, shape (n_samples,) A log-likelihood,  $p(\text{sample}|\mathbf{x})$ .

sample_given_x (x, repeat=1, **kwargs)

    x [list] This contains Theano variables, which must to correspond to ‘given’.

    repeat : int or thenao variable

    list This contains ‘x’ and sample  $\sim p(\cdot|\mathbf{x})$ , such as [x, sample].

class Tars.distributions.distribution_models.Laplace (mean_network,    var_network,
                                                    given, seed=1)
Bases: Tars.distributions.distribution_models.DistributionDouble

```

1.1.1.3 Tars.distributions.distribution_samples module

```

class Tars.distributions.distribution_samples.DeterministicSample (**kwargs)
Bases: Tars.distributions.distribution_samples.DistributionSample

Deterministic function  $p(\mathbf{x}) = f(\mathbf{x})$ 

```

log_likelihood (*sample, mean*)

sample (*mean, *args*)

mean [Theano variable, the output of a fully connected layer] (any activation function)

class Tars.distributions.distribution_samples.**BernoulliSample** (*temp=0.1, seed=1*)

Bases: *Tars.distributions.distribution_samples.GumbelSample*

Bernoulli distribution $p(x) = \text{mean}^x * (1-\text{mean})^{(1-x)}$

log_likelihood (*sample, mean*)

sample [Theano variable] This variable means test samples which you use to estimate a test log-likelihood.

mean [Theano variable, the output of a fully connected layer (Sigmoid)] This variable is a reconstruction of test samples. This must have the same shape as 'sample'.

Theano variable, shape (n_samples,) A log-likelihood, which is the same meaning as a negative binary cross-entropy error.

sample (*mean*)

mean [Theano variable, the output of a fully connected layer (Sigmoid)] The parameter (mean value) of this distribution.

Theano variable, shape (mean.shape) This variable is sampled from this distribution. i.e. $\text{sample} \sim p(x|\text{mean})$

class Tars.distributions.distribution_samples.**CategoricalSample** (*temp=0.1, seed=1*)

Bases: *Tars.distributions.distribution_samples.ConcreteSample*

Categorical distribution $p(x) = \prod \text{mean}^x$

log_likelihood (*samples, mean*)

sample [Theano variable] This variable means test samples which you use to estimate a test log-likelihood.

mean [Theano variable, the output of a fully connected layer (Softmax)] This variable is a reconstruction of test samples. This must have the same shape as 'sample'.

Theano variable, shape (n_samples,) A log-likelihood, which is the same meaning as a negative categorical cross-entropy error.

sample (*mean, onehot=True, flatten=True*)

mean [Theano variable, the output of a fully connected layer (softmax)] The parameter (mean value) of this distribution.

Theano variable, shape (mean.shape) This variable is sampled from this distribution. i.e. $\text{sample} \sim p(x|\text{mean})$

class Tars.distributions.distribution_samples.**GaussianSample** (*seed=1*)

Bases: *Tars.distributions.distribution_samples.DistributionSample*

Gaussian distribution $p(x) =$

$\frac{1}{\sqrt{2\pi\text{var}}} * \exp\{-\frac{(x-\text{mean})^2}{2\text{var}}\}$

```

log_likelihood (samples, mean, var)
    sample : Theano variable

    mean : Theano variable, the output of a fully connected layer (Linear)
    var : Theano variable, the output of a fully connected layer (Softplus)

sample (mean, var)
    mean : Theano variable, the output of a fully connected layer (Linear)
    var : Theano variable, the output of a fully connected layer (Softplus)

class Tars.distributions.distribution_samples.GaussianConstantVarSample (constant_var=1, seed=1)
    Bases: Tars.distributions.distribution_samples.GaussianSample

    log_likelihood (samples, mean)
        sample : Theano variable

        mean : Theano variable, the output of a fully connected layer (Linear)
        var : Theano variable, the output of a fully connected layer (Softplus)

    sample (mean)
        mean : Theano variable, the output of a fully connected layer (Linear)
        var : Theano variable, the output of a fully connected layer (Softplus)

class Tars.distributions.distribution_samples.LaplaceSample (seed=1, **kwargs)
    Bases: Tars.distributions.distribution_samples.DistributionSample

    Laplace distribution  $p(x) =$ 

$$\frac{1}{\sqrt{2\pi}} \exp\left\{-\frac{|x-\text{mean}|}{\sqrt{2\pi}}\right\}$$


    log_likelihood (samples, mean, b)
        sample : Theano variable

        mean : Theano variable, the output of a fully connected layer (Linear)
        b : Theano variable, the output of a fully connected layer (Softplus)

    sample (mean, b)
        mean : Theano variable, the output of a fully connected layer (Linear)
        b : Theano variable, the output of a fully connected layer (Softplus)

class Tars.distributions.distribution_samples.GumbelSample (seed=1, **kwargs)
    Bases: Tars.distributions.distribution_samples.DistributionSample

    Gumbel distribution

    log_likelihood (samples, mu, beta)
        sample : Theano variable

        mu : Theano variable, the output of a fully connected layer (Linear)
        beta : Theano variable, the output of a fully connected layer (Softplus)

    sample (mu, beta)

class Tars.distributions.distribution_samples.ConcreteSample (temp=0.1, seed=1)
    Bases: Tars.distributions.distribution_samples.GumbelSample

    Concrete distribution (Gumbel-softmax) https://arxiv.org/abs/1611.01144 https://arxiv.org/abs/1611.00712

```

```
log_likelihood()
    sample : Theano variable

    mu : Theano variable, the output of a fully connected layer (Linear)

    beta : Theano variable, the output of a fully connected layer (Softplus)

sample(mean)
    sample : Theano variable

    mean : Theano variable, the output of a fully connected layer (sigmoid or softmax)

class Tars.distributions.distribution_samples.BetaSample(iter_sampling=6, rejection_sampling=True,
                                                         seed=1)

Bases: Tars.distributions.distribution_samples.GammaSample

Beta distribution  $x^{(\alpha-1)} * (1-x)^{(\beta-1)} / B(\alpha, \beta)$ 

log_likelihood(samples, alpha, beta)

sample(alpha, beta)

class Tars.distributions.distribution_samples.DirichletSample(k,
                                                             iter_sampling=6,
                                                             rejection_sampling=True,
                                                             seed=1)

Bases: Tars.distributions.distribution_samples.GammaSample

Dirichlet distribution  $x^{(\alpha-1)} * (1-x)^{(\beta-1)} / B(\alpha, \beta)$ 

log_likelihood(samples, alpha)

sample(alpha, flatten=True)

class Tars.distributions.distribution_samples.KumaraswamySample(seed=1,
                                                              **kwargs)

Bases: Tars.distributions.distribution_samples.DistributionSample

[Naelisnick+ 2016] Deep Generative Models with Stick-Breaking Priors Kumaraswamy distribution  $p(x) = a*b*x^{(a-1)}(1-x^a)^{(b-1)}$ 

log_likelihood(samples, a, b)
    sample : Theano variable a : Theano variable, the output of a fully connected layer (Softplus) b : Theano
    variable, the output of a fully connected layer (Softplus)

sample(a, b)
    a : Theano variable, the output of a fully connected layer (Softplus) b : Theano variable, the output of a
    fully connected layer (Softplus)

class Tars.distributions.distribution_samples.UnitGaussianSample(seed=1)
    Bases: Tars.distributions.distribution_samples.GaussianSample

    Standard normal gaussian distribution  $p(x) =$ 
 $\frac{1}{\sqrt{2\pi}} * \exp\{-\frac{x^2}{2}\}$ 

log_likelihood(samples)
    sample : Theano variable

    mean : Theano variable, the output of a fully connected layer (Linear)

    var : Theano variable, the output of a fully connected layer (Softplus)

sample(shape)
```

shape [tuple] sets a shape of the output sample

```
class Tars.distributions.distribution_samples.UnitBernoulliSample (temp=0.1,  
                                                             seed=1)
```

Bases: *Tars.distributions.distribution_samples.BernoulliSample*

Unit bernoulli distribution

log_likelihood (*samples*)

sample [Theano variable] This variable means test samples which you use to estimate a test log-likelihood.

mean [Theano variable, the output of a fully connected layer (Sigmoid)] This variable is a reconstruction of test samples. This must have the same shape as ‘sample’.

Theano variable, shape (n_samples,) A log-likelihood, which is the same meaning as a negative binary cross-entropy error.

sample (*shape*)

mean [Theano variable, the output of a fully connected layer (Sigmoid)] The paramater (mean value) of this distribution.

Theano variable, shape (mean.shape) This variable is sampled from this distribution. i.e. $\text{sample} \sim p(\text{xlmean})$

```
class Tars.distributions.distribution_samples.UnitCategoricalSample (k=1,  
                                                             seed=1)
```

Bases: *Tars.distributions.distribution_samples.CategoricalSample*

Unit Categorical distribution

log_likelihood (*samples*)

sample [Theano variable] This variable means test samples which you use to estimate a test log-likelihood.

mean [Theano variable, the output of a fully connected layer (Softmax)] This variable is a reconstruction of test samples. This must have the same shape as ‘sample’.

Theano variable, shape (n_samples,) A log-likelihood, which is the same meaning as a negative categorical cross-entropy error.

sample (*shape*)

mean [Theano variable, the output of a fully connected layer (softmax)] The paramater (mean value) of this distribution.

Theano variable, shape (mean.shape) This variable is sampled from this distribution. i.e. $\text{sample} \sim p(\text{xlmean})$

```
class Tars.distributions.distribution_samples.UnitBetaSample (alpha=1.0,  
                                                             beta=1.0,  
                                                             iter_sampling=6,  
                                                             rejection_sampling=True,  
                                                             seed=1)
```

Bases: *Tars.distributions.distribution_samples.BetaSample*

Unit Beta distribution

```
log_likelihood(samples)
sample(shape)
class Tars.distributions.distribution_samples.UnitDirichletSample(k, al-
    pha=1.0,
    iter_sampling=6,
    rejection_sampling=True,
    seed=1)
Bases: Tars.distributions.distribution_samples.DirichletSample
log_likelihood(samples)
sample(shape)
```

1.1.1.4 Tars.distributions.estimate_kl module

```
Tars.distributions.estimate_kl.analytical_kl(q1, q2, given, deterministic=False)
Tars.distributions.estimate_kl.gauss_gauss_kl(mean1, var1, mean2, var2)
Tars.distributions.estimate_kl.gauss_unitgauss_kl(mean, var)
Tars.distributions.estimate_kl.gaussian_like(x, mean, var)
Tars.distributions.estimate_kl.get_prior(q)
Tars.distributions.estimate_kl.psi(b)
```

1.1.1.5 Tars.distributions.multiple module

```
class Tars.distributions.multiple.Concatenate(distributions)
Bases: object
```

This distribution is used to concatenate different distributions in their feature axis. Therefore, we can handle multiple distributions as one distribution when sampling from them or estimating their log-likelihood. However, every distribution must have same given variables.

distributions [list] Contain multiple distributions.

```
>>> from Tars.distribution import Concatenate, Gaussian, Bernoulli
>>> gauss = Gaussian(mean, var, given=[x])
>>> bernoulli = Bernoulli(mean, given=[x])
>>> concat = Concatenate([gauss, bernoulli])
```

```
fprop(x, *args, **kwargs)
```

x [list] This contains Theano variables. The number of them must be same as 'distributions'.

```
get_params()
```

```
log_likelihood_given_x(samples, **kwargs)
```

samples [list] This contains 'x', which has Theano variables, and test sample. The dimension of test sample must be same as output_dim.

Theano variable, shape (n_samples,) A log-likelihood, p(sample|x)

```
sample_given_x(x, srng, **kwargs)
```

sample_mean_given_x (*x*, **args*, ***kwargs*)

class Tars.distributions.multiple.**MultiDistributions** (*distributions*, *approximate=True*)

Bases: object

This distribution is used to stack multiple distributions, that is $p(x|z) = p(x|z_1)p(z_1|z_2)\dots p(z_{n-1}|z_n)$. If the distributions are approximate distributions, then a corresponding stacked distribution becomes like $q(z|x) = q(z_n|z_{n-1})\dots q(z_2|z_1)q(z_1|x)$. If the stacked distribution is conditioned on *y*, then the corresponding mean field approximation becomes like $p(x|z,y) = p(x|z_1)p(z_1|z_2)\dots p(z_{n-1}|z_n,y)$, or $q(z|x,y) = q(z_n|z_{n-1})\dots q(z_2|z_1)q(z_1|x,y)$. So far, each distribution except first layer cannot have conditioned variables more than two.

distributions [list] Contain multiple distributions.

```
>>> from Tars.distribution import MultiDistributions
>>> from Tars.distribution import Gaussian, Bernoulli
>>> gauss = Gaussian(mean, var, given=[x])
>>> bernoulli = Bernoulli(mean, given=[z])
>>> multi = MultiDistributions([gauss, bernoulli])
```

fprop (*x*, *layer_id=-1*, **args*, ***kwargs*)

x [list] This contains Theano variables.

mean [Theano variable] The output of this distribution.

get_params ()

log_likelihood_given_x (*samples*, ***kwargs*)

samples [list] This contains 'x', which has Theano variables, and test samples, such as z_1, z_2, \dots, z_n .

Theano variable, shape (n_samples,) **log_likelihood** (q) : $\log_q(z_1|[x,y,\dots]) + \dots + \log_q(z_n|z_{n-1})$
log_likelihood (p) : $\log_p(z_{n-1}|[z_n,y,\dots]) + \dots + \log_p(x|z_1)$

sample_given_x (*x*, *layer_id=-1*, *repeat=1*, ***kwargs*)

x [list] This contains Theano variables, which must to correspond to 'given' of first layer distribution.

repeat : int or theano variable

list This contains 'x' and samples, such as $[x, z_1, \dots, z_n]$.

sample_mean_given_x (*x*, *layer_id=-1*, **args*, ***kwargs*)

x [list] This contains Theano variables, which must to correspond to 'given'.

list This contains 'x', samples, and a mean value of sample, such as $[x, z_1, \dots, z_n, \text{mean}]$

class Tars.distributions.multiple.**MultiPriorDistributions** (*distributions*, *prior=None*)

Bases: Tars.distributions.multiple.MultiDistributions

$p(z) = p(z_n|z'_n)p(z_{n-1}|z_n, z'_n)\dots p(z_1|z_2)$.

distributions [list] Contain multiple distributions.

```
>>> from Tars.distribution import MultiPriorDistributions
>>> from Tars.distribution import Gaussian, Bernoulli
>>> gauss = Gaussian(mean, var, given=[z2])
>>> bernoulli = Bernoulli(mean, given=[z1])
>>> multi = MultiPriorDistributions([gauss, bernoulli])
```

log_likelihood_given_x (*samples*, *add_prior=True*, ***kwargs*)

samples [list] This contains 'x', which has Theano variables, and test samples, such as z_1, z_2, \dots, z_n .

Theano variable, shape (n_samples,)

log_likelihood : *add_prior=True* : $\log_p(z_n, z^n) + \log_p(z_{n-1} | z^n, z^n) + \dots + \log_p(z_2 | z_1)$
 add_prior=False : $\log_p(z_{n-1} | z^n, z^n) + \dots + \log_p(z_2 | z_1)$

1.1.1.6 Module contents

class Tars.distributions.**Distribution** (*distribution*, *mean_network*, *given*, *seed=1*,
 set_log_likelihood=True)

Bases: object

mean_network [lasagne.layers.Layer] The network whose outputs express the parameter of this distribution.

given [list] This contains instances of lasagne.layers.InputLayer, which mean the conditioning variables. e.g. if *given* = [x,y], then the corresponding log-likelihood is

$\log p(*|x, y)$

fprop (*x*, **args*, ***kwargs*)

x [list] This contains Theano variables, which must correspond to 'given'.

mean [Theano variable] The output of this distribution.

get_input_shape ()

tuple This represents the shape of the inputs of this distribution.

get_output_shape ()

tuple This represents the shape of the output of this distribution.

get_params ()

log_likelihood_given_x (*samples*, ***kwargs*)

samples [list] This contains 'x', which has Theano variables, and test sample.

Theano variable, shape (n_samples,) A log-likelihood, $p(\text{sample}|x)$.

sample_given_x (*x*, *repeat=1*, ***kwargs*)

x [list] This contains Theano variables, which must correspond to 'given'.

repeat : int or theano variable

list This contains 'x' and sample $\sim p(*|x)$, such as [x, sample].

sample_mean_given_x (*x*, **args*, ***kwargs*)

x [list] This contains Theano variables, which must correspond to 'given'.

list This contains 'x' and a mean value of sample $\sim p(*|x)$.

set_seed (*seed=1*)

class Tars.distributions.**Deterministic** (*network*, *given*, *seed=1*)

Bases: Tars.distributions.distribution_models.Distribution

```

class Tars.distributions.Bernoulli (mean_network, given, temp=0.1, seed=1)
    Bases: Tars.distributions.distribution_models.Distribution

class Tars.distributions.Categorical (mean_network, given, temp=0.1, n_dim=1, seed=1)
    Bases: Tars.distributions.distribution_models.Distribution

fprop (x, *args, **kwargs)

    x [list] This contains Theano variables, which must to correspond to ‘given’.

    mean [Theano variable] The output of this distribution.

sample_given_x (x, repeat=1, **kwargs)

    x [list] This contains Theano variables, which must to correspond to ‘given’.
    repeat : int or thenao variable
    list This contains ‘x’ and sample  $\sim p(*|x)$ , such as [x, sample].

class Tars.distributions.Gaussian (mean_network, var_network, given, seed=1)
    Bases: Tars.distributions.distribution_models.DistributionDouble

class Tars.distributions.GaussianConstantVar (mean_network, given, var=1, seed=1)
    Bases: Tars.distributions.distribution_models.Distribution

class Tars.distributions.Laplace (mean_network, var_network, given, seed=1)
    Bases: Tars.distributions.distribution_models.DistributionDouble

class Tars.distributions.Kumaraswamy (a_network, b_network, given, stick_breaking=True, seed=1)
    Bases: Tars.distributions.distribution_models.DistributionDouble
    [Naelisnick+ 2016] Deep Generative Models with Stick-Breaking Priors

log_likelihood_given_x (*args)

    samples [list] This contains ‘x’, which has Theano variables, and test sample.

    Theano variable, shape (n_samples,) A log-likelihood,  $p(\text{sample}|x)$ .

sample_given_x (x, repeat=1, **kwargs)

    x [list] This contains Theano variables, which must to correspond to ‘given’.
    repeat : int or thenao variable
    list This contains ‘x’ and sample  $\sim p(*|x)$ , such as [x, sample].

class Tars.distributions.Gamma (alpha_network, beta_network, given, seed=1)
    Bases: Tars.distributions.distribution_models.DistributionDouble

class Tars.distributions.Beta (alpha_network, beta_network, given, iter_sampling=6, rejection_sampling=True, seed=1)
    Bases: Tars.distributions.distribution_models.DistributionDouble

class Tars.distributions.Dirichlet (alpha_network, given, k, iter_sampling=6, rejection_sampling=True, seed=1)
    Bases: Tars.distributions.distribution_models.Distribution

sample_given_x (x, repeat=1, **kwargs)

    x [list] This contains Theano variables, which must to correspond to ‘given’.
    repeat : int or thenao variable

```

list This contains 'x' and sample $\sim p(*|x)$, such as [x, sample].

class Tars.distributions.Concatenate (*distributions*)

Bases: object

This distribution is used to concatenate different distributions in their feature axis. Therefore, we can handle multiple distributions as one distribution when sampling from them or estimating their log-likelihood. However, every distribution must have same given variables.

distributions [list] Contain multiple distributions.

```
>>> from Tars.distribution import Concatenate, Gaussian, Bernoulli
>>> gauss = Gaussian(mean, var, given=[x])
>>> bernoulli = Bernoulli(mean, given=[x])
>>> concat = Concatenate([gauss, bernoulli])
```

fprop (*x*, *args, **kwargs)

x [list] This contains Theano variables. The number of them must be same as 'distributions'.

get_params ()

log_likelihood_given_x (*samples*, **kwargs)

samples [list] This contains 'x', which has Theano variables, and test sample. The dimension of test sample must be same as output_dim.

Theano variable, shape (n_samples,) A log-likelihood, $p(\text{sample}|x)$

sample_given_x (*x*, *srng*, **kwargs)

sample_mean_given_x (*x*, *args, **kwargs)

class Tars.distributions.MultiDistributions (*distributions*, *approximate=True*)

Bases: object

This distribution is used to stack multiple distributions, that is $p(x|z) = p(x|z_1)p(z_1|z_2) \dots p(z_{n-1}|z_n)$. If the distributions are approximate distributions, then a corresponding stacked distribution becomes like $q(z|x) = q(z_n|z_{n-1}) \dots q(z_2|z_1)q(z_1|x)$. If the stacked distribution is conditioned on y, then the corresponding mean field approximation becomes like $p(x|z,y) = p(x|z_1)p(z_1|z_2) \dots p(z_{n-1}|z_n,y)$, or $q(z|x,y) = q(z_n|z_{n-1}) \dots q(z_2|z_1)q(z_1|x,y)$. So far, each distribution except first layer cannot have conditioned variables more than two.

distributions [list] Contain multiple distributions.

```
>>> from Tars.distribution import MultiDistributions
>>> from Tars.distribution import Gaussian, Bernoulli
>>> gauss = Gaussian(mean, var, given=[x])
>>> bernoulli = Bernoulli(mean, given=[z])
>>> multi = MultiDistributions([gauss, bernoulli])
```

fprop (*x*, *layer_id=-1*, *args, **kwargs)

x [list] This contains Theano variables.

mean [Theano variable] The output of this distribution.

get_params ()

log_likelihood_given_x (*samples*, **kwargs)

samples [list] This contains 'x', which has Theano variables, and test samples, such as z_1, z_2, \dots, z_n .

Theano variable, shape (n_samples,) `log_likelihood (q) :` $\log_q(z_1|[x,y,\dots])+ \dots + \log_q(z_n|z_{n-1})$
`log_likelihood (p) :` $\log_p(z_{n-1}|[z_n,y,\dots])+ \dots + \log_p(x|z_1)$

sample_given_x (*x*, *layer_id=-1*, *repeat=1*, ***kwargs*)

x [list] This contains Theano variables, which must to correspond to ‘given’ of first layer distribution.

repeat : int or theano variable

list This contains ‘x’ and samples, such as $[x, z_1, \dots, z_n]$.

sample_mean_given_x (*x*, *layer_id=-1*, **args*, ***kwargs*)

x [list] This contains Theano variables, which must to correspond to ‘given’.

list This contains ‘x’, samples, and a mean value of sample, such as $[x, z_1, \dots, z_n, \text{mean}]$

class `Tars.distributions.MultiPriorDistributions` (*distributions*, *prior=None*)

Bases: `Tars.distributions.multiple.MultiDistributions`

$p(z) = p(z_n, z'n)p(z_{n-1}|z_n, z'n) \dots p(z_1|z_2)$.

distributions [list] Contain multiple distributions.

```
>>> from Tars.distribution import MultiPriorDistributions
>>> from Tars.distribution import Gaussian, Bernoulli
>>> gauss = Gaussian(mean, var, given=[z2])
>>> bernoulli = Bernoulli(mean, given=[z1])
>>> multi = MultiPriorDistributions([gauss, bernoulli])
```

log_likelihood_given_x (*samples*, *add_prior=True*, ***kwargs*)

samples [list] This contains ‘x’, which has Theano variables, and test samples, such as z_1, z_2, \dots, z_n .

Theano variable, shape (n_samples,)

log_likelihood : `add_prior=True` : $\log_p(z_n, z'n) + \log_p(z_{n-1}|z_n, z'n) + \dots + \log_p(z_2|z_1)$
`add_prior=False` : $\log_p(z_{n-1}|z_n, z'n) + \dots + \log_p(z_2|z_1)$

1.1.2 Tars.layers package

1.1.2.1 Submodules

1.1.2.2 Tars.layers.conv_recurrent module

1.1.2.3 Tars.layers.recurrent module

1.1.2.4 Tars.layers.shape module

1.1.2.5 Module contents

1.1.3 Tars.models package

1.1.3.1 Submodules

1.1.3.2 Tars.models.ae module

```
class Tars.models.ae.AE(q, p, n_batch=100, optimizer=<function adam>, optimizer_params={},
                        clip_grad=None, max_norm_constraint=None, seed=1234)
    Bases: Tars.models.model.Model

    test (test_set, n_batch=None, verbose=True)

    train (train_set, verbose=False)
```

1.1.3.3 Tars.models.gan module

```
class Tars.models.gan.GAN(p, d, n_batch=100, p_optimizer=<function adam>,
                           d_optimizer=<function adam>, p_optimizer_params={},
                           d_optimizer_params={}, p_critic=<function <lambda>>,
                           d_critic=<function <lambda>>, p_clip_param=None,
                           d_clip_param=None, p_clip_grad=None, d_clip_grad=None,
                           p_max_norm_constraint=None, d_max_norm_constraint=None,
                           ll_lambda=0, seed=1234)
    Bases: Tars.models.model.Model

    gan_test (test_set, n_batch=None, verbose=False)

    train (train_set, freq=1, verbose=False)
```

1.1.3.4 Tars.models.model module

```
class Tars.models.model.Model(n_batch=100, seed=1234)
    Bases: object

    set_seed (seed=1234)

    train ()
```

1.1.3.5 Tars.models.vae module

```
class Tars.models.vae.VAE(q, p, prior=None, n_batch=100, optimizer=<function adam>,
                        optimizer_params={}, clip_grad=None, max_norm_constraint=None,
                        train_iw=False, test_iw=True, iw_alpha=0, seed=1234)
    Bases: Tars.models.model.Model

    test (test_set, l=1, k=1, n_batch=None, verbose=True)

    train (train_set, l=1, k=1, annealing_beta=1, verbose=False)
```

1.1.3.6 Module contents

```
class Tars.models.AE(q, p, n_batch=100, optimizer=<function adam>, optimizer_params={},
                    clip_grad=None, max_norm_constraint=None, seed=1234)
    Bases: Tars.models.model.Model

    test (test_set, n_batch=None, verbose=True)

    train (train_set, verbose=False)
```

```
class Tars.models.VAE(q, p, prior=None, n_batch=100, optimizer=<function adam>, op-
                    timizer_params={}, clip_grad=None, max_norm_constraint=None,
                    train_iw=False, test_iw=True, iw_alpha=0, seed=1234)
    Bases: Tars.models.model.Model

    test (test_set, l=1, k=1, n_batch=None, verbose=True)

    train (train_set, l=1, k=1, annealing_beta=1, verbose=False)
```

```
class Tars.models.GAN(p, d, n_batch=100, p_optimizer=<function adam>,
                    d_optimizer=<function adam>, p_optimizer_params={},
                    d_optimizer_params={}, p_critic=<function <lambda>>,
                    d_critic=<function <lambda>>, p_clip_param=None, d_clip_param=None,
                    p_clip_grad=None, d_clip_grad=None, p_max_norm_constraint=None,
                    d_max_norm_constraint=None, ll_lambda=0, seed=1234)
    Bases: Tars.models.model.Model

    gan_test (test_set, n_batch=None, verbose=False)

    train (train_set, freq=1, verbose=False)
```

```
class Tars.models.JMVAE(q, p, prior=None, n_batch=100, optimizer=<function adam>, op-
                    timizer_params={}, clip_grad=None, max_norm_constraint=None,
                    train_iw=False, test_iw=True, seed=1234)
    Bases: Tars.models.vae.VAE

    test (test_set, l=1, k=1, index=[0], sampling_n=1, missing_resample=False, type_p='joint', miss-
                    ing=False, n_batch=None, verbose=True)
```

```
class Tars.models.JMVAE_KL(q, p, pseudo_q, prior=None, gamma=1, n_batch=100, opti-
                    mizer=<function adam>, optimizer_params={}, clip_grad=None,
                    max_norm_constraint=None, test_iw=True, seed=1234)
    Bases: Tars.models.jmvae.JMVAE
```

```
class Tars.models.CMMA(q, p, n_batch=100, optimizer=<function adam>, optimizer_params={},
                    clip_grad=None, max_norm_constraint=None, train_iw=False,
                    test_iw=True, iw_alpha=0, seed=1234)
    Bases: Tars.models.vae.VAE

    test (test_set, l=1, k=1, type_p='normal', missing=False, n_batch=None, verbose=True)
```

```
class Tars.models.CVAE(q, p, prior=None, n_batch=100, optimizer=<function adam>, op-  
                      timizer_params={}, clip_grad=None, max_norm_constraint=None,  
                      train_iw=False, test_iw=True, iw_alpha=0, seed=1234)  
    Bases: Tars.models.vae.VAE  
  
    test (test_set, l=1, k=1, n_batch=None, verbose=True, type_p='normal', missing=False)
```

1.1.4 Tars.tests package

1.1.4.1 Submodules

1.1.4.2 Tars.tests.test_utils module

```
class Tars.tests.test_utils.TestEpsilon(methodName='runTest')  
    Bases: unittest.case.TestCase  
  
    test_it()  
  
class Tars.tests.test_utils.TestLogMeanExp(methodName='runTest')  
    Bases: unittest.case.TestCase  
  
    test_1d()  
  
    test_2d()  
  
class Tars.tests.test_utils.TestLogSumExp(methodName='runTest')  
    Bases: unittest.case.TestCase  
  
    test_1d()  
  
    test_2d()  
  
class Tars.tests.test_utils.TestToList(methodName='runTest')  
    Bases: unittest.case.TestCase  
  
    test_it()
```

1.1.4.3 Module contents

1.2 Submodules

1.3 Tars.load_data module

```
Tars.load_data.celeba(datapath, toFloat=True, gray=False, rate=0.001, rseed=0)  
Tars.load_data.facade(datapath)  
Tars.load_data.flickr(datapath, toFloat=True)  
Tars.load_data.lfw(datapath, toFloat=True, gray=False, rate=0.1, rseed=0)  
Tars.load_data.mnist(datapath, toFloat=False)  
Tars.load_data.one_of_k(a)  
  
class Tars.load_data.paramaters(mean=0, std=0)  
Tars.load_data.svh(datapath, toFloat=True, binarize_y=True, gray=False, extra=True)
```

1.4 Tars.utils module

```
Tars.utils.epsilon()  
Tars.utils.load_weights(network, name)  
Tars.utils.log_mean_exp(x, axis=0, keepdims=False)  
Tars.utils.log_sum_exp(x, axis=0, keepdims=False)  
Tars.utils.save_weights(network, name)  
Tars.utils.set_epsilon(eps)  
Tars.utils.tolist(x)
```

1.5 Module contents

CHAPTER 2

Indices and tables

- `genindex`
- `modindex`
- `search`

t

- Tars, [19](#)
- Tars.distributions, [12](#)
- Tars.distributions.distribution_models,
[3](#)
- Tars.distributions.distribution_samples,
[5](#)
- Tars.distributions.estimate_kl, [10](#)
- Tars.distributions.multiple, [10](#)
- Tars.load_data, [18](#)
- Tars.models, [17](#)
- Tars.models.ae, [16](#)
- Tars.models.gan, [16](#)
- Tars.models.model, [16](#)
- Tars.models.vae, [17](#)
- Tars.tests, [18](#)
- Tars.tests.test_utils, [18](#)
- Tars.utils, [19](#)

A

AE (class in Tars.models), 17
 AE (class in Tars.models.ae), 16
 analytical_kl() (in module Tars.distributions.estimate_kl), 10

B

Bernoulli (class in Tars.distributions), 12
 Bernoulli (class in Tars.distributions.distribution_models), 3
 BernoulliSample (class in Tars.distributions.distribution_samples), 6
 Beta (class in Tars.distributions), 13
 Beta (class in Tars.distributions.distribution_models), 3
 BetaSample (class in Tars.distributions.distribution_samples), 8

C

Categorical (class in Tars.distributions), 13
 Categorical (class in Tars.distributions.distribution_models), 3
 CategoricalSample (class in Tars.distributions.distribution_samples), 6
 celeba() (in module Tars.load_data), 18
 CMMA (class in Tars.models), 17
 Concatenate (class in Tars.distributions), 14
 Concatenate (class in Tars.distributions.multiple), 10
 ConcreteSample (class in Tars.distributions.distribution_samples), 7
 CVAE (class in Tars.models), 17

D

Deterministic (class in Tars.distributions), 12
 Deterministic (class in Tars.distributions.distribution_models), 3
 DeterministicSample (class in Tars.distributions.distribution_samples), 5
 Dirichlet (class in Tars.distributions), 13
 Dirichlet (class in Tars.distributions.distribution_models), 4

DirichletSample (class in Tars.distributions.distribution_samples), 8
 Distribution (class in Tars.distributions), 12
 Distribution (class in Tars.distributions.distribution_models), 4
 DistributionDouble (class in Tars.distributions.distribution_models), 5

E

epsilon() (in module Tars.utils), 19

F

facade() (in module Tars.load_data), 18
 flickr() (in module Tars.load_data), 18
 fprop() (Tars.distributions.Categorical method), 13
 fprop() (Tars.distributions.Concatenate method), 14
 fprop() (Tars.distributions.Distribution method), 12
 fprop() (Tars.distributions.distribution_models.Categorical method), 3
 fprop() (Tars.distributions.distribution_models.Distribution method), 4
 fprop() (Tars.distributions.distribution_models.DistributionDouble method), 5
 fprop() (Tars.distributions.MultiDistributions method), 14
 fprop() (Tars.distributions.multiple.Concatenate method), 10
 fprop() (Tars.distributions.multiple.MultiDistributions method), 11

G

Gamma (class in Tars.distributions), 13
 Gamma (class in Tars.distributions.distribution_models), 5
 GAN (class in Tars.models), 17
 GAN (class in Tars.models.gan), 16
 gan_test() (Tars.models.GAN method), 17
 gan_test() (Tars.models.gan.GAN method), 16
 gauss_gauss_kl() (in module Tars.distributions.estimate_kl), 10

gauss_unitgauss_kl() (in module Tars.distributions.estimate_kl), 10
 Gaussian (class in Tars.distributions), 13
 Gaussian (class in Tars.distributions.distribution_models), 5
 gaussian_like() (in module Tars.distributions.estimate_kl), 10
 GaussianConstantVar (class in Tars.distributions), 13
 GaussianConstantVar (class in Tars.distributions.distribution_models), 5
 GaussianConstantVarSample (class in Tars.distributions.distribution_samples), 7
 GaussianSample (class in Tars.distributions.distribution_samples), 6
 get_input_shape() (Tars.distributions.Distribution method), 12
 get_input_shape() (Tars.distributions.distribution_models.Distribution method), 4
 get_output_shape() (Tars.distributions.Distribution method), 12
 get_output_shape() (Tars.distributions.distribution_models.Distribution method), 4
 get_params() (Tars.distributions.Concatenate method), 14
 get_params() (Tars.distributions.Distribution method), 12
 get_params() (Tars.distributions.distribution_models.Distribution method), 4
 get_params() (Tars.distributions.distribution_models.Distribution method), 5
 get_params() (Tars.distributions.MultiDistributions method), 14
 get_params() (Tars.distributions.multiple.Concatenate method), 10
 get_params() (Tars.distributions.multiple.MultiDistributions method), 11
 get_prior() (in module Tars.distributions.estimate_kl), 10
 GumbelSample (class in Tars.distributions.distribution_samples), 7

J

JMVAE (class in Tars.models), 17
 JMVAE_KL (class in Tars.models), 17

K

Kumaraswamy (class in Tars.distributions), 13
 Kumaraswamy (class in Tars.distributions.distribution_models), 5
 KumaraswamySample (class in Tars.distributions.distribution_samples), 8

L

Laplace (class in Tars.distributions), 13
 Laplace (class in Tars.distributions.distribution_models), 5

LaplaceSample (class in Tars.distributions.distribution_samples), 7
 lfw() (in module Tars.load_data), 18
 load_weights() (in module Tars.utils), 19
 log_likelihood() (Tars.distributions.distribution_samples.BernoulliSample method), 6
 log_likelihood() (Tars.distributions.distribution_samples.BetaSample method), 8
 log_likelihood() (Tars.distributions.distribution_samples.CategoricalSample method), 6
 log_likelihood() (Tars.distributions.distribution_samples.ConcreteSample method), 7
 log_likelihood() (Tars.distributions.distribution_samples.DeterministicSample method), 5
 log_likelihood() (Tars.distributions.distribution_samples.DirichletSample method), 8
 log_likelihood() (Tars.distributions.distribution_samples.GaussianConstantVar method), 7
 log_likelihood() (Tars.distributions.distribution_samples.GaussianSample method), 6
 log_likelihood() (Tars.distributions.distribution_samples.GumbelSample method), 7
 log_likelihood() (Tars.distributions.distribution_samples.KumaraswamySample method), 8
 log_likelihood() (Tars.distributions.distribution_samples.LaplaceSample method), 7
 log_likelihood() (Tars.distributions.distribution_samples.UnitBernoulliSample method), 9
 log_likelihood() (Tars.distributions.distribution_samples.UnitBetaSample method), 9
 log_likelihood() (Tars.distributions.distribution_samples.UnitCategoricalSample method), 9
 log_likelihood() (Tars.distributions.distribution_samples.UnitDirichletSample method), 10
 log_likelihood() (Tars.distributions.distribution_samples.UnitGaussianSample method), 8
 log_likelihood_given_x() (Tars.distributions.Concatenate method), 14
 log_likelihood_given_x() (Tars.distributions.Distribution method), 12
 log_likelihood_given_x() (Tars.distributions.distribution_models.Distribution method), 4
 log_likelihood_given_x() (Tars.distributions.distribution_models.Kumaraswamy method), 5
 log_likelihood_given_x() (Tars.distributions.Kumaraswamy method), 13
 log_likelihood_given_x() (Tars.distributions.MultiDistributions method), 14
 log_likelihood_given_x() (Tars.distributions.multiple.Concatenate method), 10

method), 10
 log_likelihood_given_x()
 (Tars.distributions.multiple.MultiDistributions
 method), 11
 log_likelihood_given_x()
 (Tars.distributions.multiple.MultiPriorDistributions
 method), 11
 log_likelihood_given_x()
 (Tars.distributions.MultiPriorDistributions
 method), 15
 log_mean_exp() (in module Tars.utils), 19
 log_sum_exp() (in module Tars.utils), 19

M

mnist() (in module Tars.load_data), 18
 Model (class in Tars.models.model), 16
 MultiDistributions (class in Tars.distributions), 14
 MultiDistributions (class in Tars.distributions.multiple),
 11
 MultiPriorDistributions (class in Tars.distributions), 15
 MultiPriorDistributions (class in
 Tars.distributions.multiple), 11

O

one_of_k() (in module Tars.load_data), 18

P

paramaters (class in Tars.load_data), 18
 psi() (in module Tars.distributions.estimate_kl), 10

S

sample() (Tars.distributions.distribution_samples.BernoulliSample
 method), 6
 sample() (Tars.distributions.distribution_samples.BetaSample
 method), 8
 sample() (Tars.distributions.distribution_samples.CategoricalSample
 method), 6
 sample() (Tars.distributions.distribution_samples.ConcreteSample
 method), 8
 sample() (Tars.distributions.distribution_samples.DeterministicSample
 method), 6
 sample() (Tars.distributions.distribution_samples.DirichletSample
 method), 8
 sample() (Tars.distributions.distribution_samples.GaussianConstantVarSample
 method), 7
 sample() (Tars.distributions.distribution_samples.GaussianSample
 method), 7
 sample() (Tars.distributions.distribution_samples.GumbelSample
 method), 7
 sample() (Tars.distributions.distribution_samples.KumaraswamySample
 method), 8
 sample() (Tars.distributions.distribution_samples.LaplaceSample
 method), 7
 sample() (Tars.distributions.distribution_samples.UnitBernoulliSample
 method), 9
 sample() (Tars.distributions.distribution_samples.UnitBetaSample
 method), 10
 sample() (Tars.distributions.distribution_samples.UnitCategoricalSample
 method), 9
 sample() (Tars.distributions.distribution_samples.UnitDirichletSample
 method), 10
 sample() (Tars.distributions.distribution_samples.UnitGaussianSample
 method), 8
 sample_given_x() (Tars.distributions.Categorical
 method), 13
 sample_given_x() (Tars.distributions.Concatenate
 method), 14
 sample_given_x() (Tars.distributions.Dirichlet method),
 13
 sample_given_x() (Tars.distributions.Distribution
 method), 12
 sample_given_x() (Tars.distributions.distribution_models.Categorical
 method), 3
 sample_given_x() (Tars.distributions.distribution_models.Dirichlet
 method), 4
 sample_given_x() (Tars.distributions.distribution_models.Distribution
 method), 4
 sample_given_x() (Tars.distributions.distribution_models.Kumaraswamy
 method), 5
 sample_given_x() (Tars.distributions.Kumaraswamy
 method), 13
 sample_given_x() (Tars.distributions.MultiDistributions
 method), 15
 sample_given_x() (Tars.distributions.multiple.Concatenate
 method), 10
 sample_given_x() (Tars.distributions.multiple.MultiDistributions
 method), 11
 sample_mean_given_x() (Tars.distributions.Concatenate
 method), 14
 sample_mean_given_x() (Tars.distributions.Distribution
 method), 12
 sample_mean_given_x() (Tars.distributions.distribution_models.Distribution
 method), 4
 sample_mean_given_x() (Tars.distributions.MultiDistributions
 method), 15
 sample_mean_given_x() (Tars.distributions.multiple.Concatenate
 method), 10
 sample_mean_given_x() (Tars.distributions.multiple.MultiDistributions
 method), 11
 save_weights() (in module Tars.utils), 19
 set_epsilon() (in module Tars.utils), 19
 set_seed() (Tars.distributions.Distribution method), 12
 set_seed() (Tars.distributions.distribution_models.Distribution
 method), 5
 set_seed() (Tars.models.model.Model method), 16
 split() (in module Tars.load_data), 18

T

Tars (module), [19](#)
Tars.distributions (module), [12](#)
Tars.distributions.distribution_models (module), [3](#)
Tars.distributions.distribution_samples (module), [5](#)
Tars.distributions.estimate_kl (module), [10](#)
Tars.distributions.multiple (module), [10](#)
Tars.load_data (module), [18](#)
Tars.models (module), [17](#)
Tars.models.ae (module), [16](#)
Tars.models.gan (module), [16](#)
Tars.models.model (module), [16](#)
Tars.models.vae (module), [17](#)
Tars.tests (module), [18](#)
Tars.tests.test_utils (module), [18](#)
Tars.utils (module), [19](#)
test() (Tars.models.AE method), [17](#)
test() (Tars.models.ae.AE method), [16](#)
test() (Tars.models.CMMA method), [17](#)
test() (Tars.models.CVAE method), [18](#)
test() (Tars.models.JMVAE method), [17](#)
test() (Tars.models.VAE method), [17](#)
test() (Tars.models.vae.VAE method), [17](#)
test_1d() (Tars.tests.test_utils.TestLogMeanExp method), [18](#)
test_1d() (Tars.tests.test_utils.TestLogSumExp method), [18](#)
test_2d() (Tars.tests.test_utils.TestLogMeanExp method), [18](#)
test_2d() (Tars.tests.test_utils.TestLogSumExp method), [18](#)
test_it() (Tars.tests.test_utils.TestEpsilon method), [18](#)
test_it() (Tars.tests.test_utils.TestToList method), [18](#)
TestEpsilon (class in Tars.tests.test_utils), [18](#)
TestLogMeanExp (class in Tars.tests.test_utils), [18](#)
TestLogSumExp (class in Tars.tests.test_utils), [18](#)
TestToList (class in Tars.tests.test_utils), [18](#)
tolist() (in module Tars.utils), [19](#)
train() (Tars.models.AE method), [17](#)
train() (Tars.models.ae.AE method), [16](#)
train() (Tars.models.GAN method), [17](#)
train() (Tars.models.gan.GAN method), [16](#)
train() (Tars.models.model.Model method), [16](#)
train() (Tars.models.VAE method), [17](#)
train() (Tars.models.vae.VAE method), [17](#)

U

UnitBernoulliSample (class in Tars.distributions.distribution_samples), [9](#)
UnitBetaSample (class in Tars.distributions.distribution_samples), [9](#)
UnitCategoricalSample (class in Tars.distributions.distribution_samples), [9](#)

UnitDirichletSample (class in Tars.distributions.distribution_samples), [10](#)
UnitGaussianSample (class in Tars.distributions.distribution_samples), [8](#)

V

VAE (class in Tars.models), [17](#)
VAE (class in Tars.models.vae), [17](#)