Tars Documentation

Release 0.0.2

Author

Jul 05, 2018

Contents

Contents:

CHAPTER 1

Tars package

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 thenao variable

list This contains 'x' and sample ~ 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*, *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 ~ p(*|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-likehood is

log p(*|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 ~ p(*|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 ~ p(*|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(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 ~ p(*|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(x) = f(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) = mean^x * (1-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 paramater (mean value) of this distribution.

Theano variable, shape (mean.shape) This variable is sampled from this distribution. i.e. sample ~ p(x|mean)

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

Bases: *Tars.distributions.distribution_samples.ConcreteSample*

Categorical distribution p(x) = prod 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 paramater (mean value) of this distribution.

Theano variable, shape (mean.shape) This variable is sampled from this distribution. i.e. sample ~ p(x|mean)

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

Bases: Tars.distributions.distribution_samples.DistributionSample

Gaussian distribution p(x) =

rac{1}{sqrt{2*pi*var}} * exp{- rac{{x-mean}^2}{2*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) =

rac{1}{sqrt{2*phi}} * exp{- rac{**|x-mean|**}{phi}}

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) =

rac{1}{sqrt{2*pi}} * exp{- rac{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. sample ~ p(x|mean)

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. sample ~ p(x|mean)

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*, *alpha=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 distiributions, that is p(x|z) = p(x|z1)p(z1|z2). . . p(zn-1|zn). If the distributions are approximate distributions, then a corresponding stacked distribution becomes like q(z|x) = q(zn|zn-1). . . q(z2|z1)q(z1|x). If the stacked distribution is conditioned on y, then the corresponding mean field approximation becomes like p(x|z,y) = p(x|z1)p(z1|z2). . . p(zn-1|zn,y), or q(z|x,y) = q(zn|zn-1). . . q(z2|z1)q(z1|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 z1, z2,. . . ,zn.

Theano variable, shape (n_samples,) log_likelihood (q) : log_q(z1|[x,y,. . .])+. . . +log_q(zn|zn-1) log_likelihood (p) : log_p(zn-1|[zn,y,. . .])+. . . +log_p(x|z1)

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 distibution.

repeat : int or thenao variable

list This contains 'x' and samples, such as [x,z1,. . . ,zn].

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,z1,. . . ,zn_mean]

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

Bases: *Tars.distributions.multiple.MultiDistributions*

p(z) = p(zn,z'n)p(zn-1|zn,z'n). . . p(z1|z2).

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 z1, z2,. . . ,zn.

Theano variable, shape (n_samples,)

log_likelihood : add_prior=True : log_p(zn,z'n)+log_p(zn-1|zn,z'n)+. . . +log_p(z2|z1) add_prior=False : log_p(zn-1|zn,z'n)+. . . +log_p(z2|z1)

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 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-likehood is

log p(*|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 ~ p(*|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 ~ 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 ~ 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(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 ~ 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 ~ 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(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 distiributions, that is p(x|z) = p(x|z1)p(z1|z2)...p(zn-1|zn). If the distributions are approximate distributions, then a corresponding stacked distribution becomes like q(z|x) = q(zn|zn-1)...q(z2|z1)q(z1|x). If the stacked distribution is conditioned on y, then the corresponding mean field approximation becomes like p(x|z,y) = p(x|z1)p(z1|z2)...p(zn-1|zn,y), or q(z|x,y) = q(zn|zn-1)...q(z2|z1)q(z1|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 z1, z2,...,zn.

Theano variable, shape (n_samples,) log_likelihood (q) : log_q(z1|[x,y,. . .])+. . . +log_q(zn|zn-1)
log_likelihood (p) : log_p(zn-1|[zn,y,. . .])+. . . +log_p(x|z1)

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 distibution.

repeat : int or thenao variable

list This contains 'x' and samples, such as [x,z1,. . . ,zn].

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,z1,. . . ,zn_mean]

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

Bases: *Tars.distributions.multiple.MultiDistributions*

p(z) = p(zn,z'n)p(zn-1|zn,z'n). . . p(z1|z2).

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 z1, z2,. . . ,zn.

Theano variable, shape (n_samples,)

log_likelihood : add_prior=True : log_p(zn,z'n)+log_p(zn-1|zn,z'n)+. . . +log_p(z2|z1)
add_prior=False : log_p(zn-1|zn,z'n)+. . . +log_p(z2|z1)

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*, *l1_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>, 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*)

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, l1_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>, optimizer_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', missing=False, n_batch=None, verbose=True*)

class Tars.models.**JMVAE_KL**(*q, p, pseudo_q, prior=None, gamma=1, n_batch=100, optimizer=<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>*, *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*, *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.**svhn**(*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

Python Module Index

t

Index

J

K

L

T

U

V