Pixyz Documentation

masa

Jan 19, 2019

Package Reference

Pixyz is a library for developing deep generative models in a more concise, intuitive and extendable way!

CHAPTER 1

pixyz.distributions (Distribution API)

1.1 Distribution

class pixyz.distributions.distributions.**Distribution**(*cond_var=[], var=['x'], name='p', dim=1*)

Bases: torch.nn.modules.module.Module

Distribution class. In pixyz, all distributions are required to inherit this class.

var [list] Variables of this distribution.

cond_var [list] Conditional variables of this distribution. In case that cond_var is not empty, we must set the corresponding inputs in order to sample variables or estimate the log likelihood.

dim [int] Number of dimensions of this distribution. This might be ignored depending on the shape which is set in the sample method and on its parent distribution. Moreover, this is not consider when this class is inherited by DNNs. This is set to 1 by default.

name [str] Name of this distribution. This name is displayed in prob_text and prob_factorized_text. This is set to "p" by default.

distribution_name

name

var

cond_var

input_var
Normally, *input_var* has same values as *cond_var*.

prob_text

prob_factorized_text

get_params(*params_dict={}*)
This method aims to get parameters of this distributions from constant parameters set in initialization and outputs of DNNs.

params_dict [dict] Input parameters.

output_dict [dict] Output parameters

```
>>> print(dist_1.prob_text, dist_1.distribution_name)
p(x) Normal
>>> dist_1.get_params()
{"loc": 0, "scale": 1}
>>> print(dist_2.prob_text, dist_2.distribution_name)
p(x|z) Normal
>>> dist_1.get_params({"z": 1})
{"loc": 0, "scale": 1}
```

sample(*x={}*, *shape=None*, *batch_size=1*, *return_all=True*, *reparam=False*)
Sample variables of this distribution. If *cond_var* is not empty, we should set inputs as a dictionary format.

x [torch.Tensor, list, or dict] Input variables.

shape [tuple] Shape of samples. If set, *batch_size* and *dim* are ignored.

batch_size [int] Batch size of samples. This is set to 1 by default.

return_all [bool] Choose whether the output contains input variables.

reparam [bool] Choose whether we sample variables with reparameterized trick.

output [dict] Samples of this distribution.

log_likelihood(*x_dict*)
Estimate the log likelihood of this distribution from inputs formatted by a dictionary.

x_dict [dict] Input samples.

log_like [torch.Tensor] Log-likelihood.

forward(**args*, ***kwargs*)
When this class is inherited by DNNs, it is also intended that this method is overrided.

sample_mean(*x*)

replace_var(***replace_dict*)

marginalize_var(*marginalize_list*)

1.2 Exponential families

1.2.1 Normal

class pixyz.distributions.**Normal**(*cond_var=[], var=['x'], name='p', dim=None, **kwargs*)
Bases: pixyz.distributions.distributions.DistributionBase

distribution_name

sample_mean(*x*)

1.2.2 Bernoulli

class pixyz.distributions.**Bernoulli**(*cond_var=[], var=['x'], name='p', dim=None, **kwargs*)

Bases: pixyz.distributions.distributions.DistributionBase

distribution_name

sample_mean(*x*)

1.2.3 RelaxedBernoulli

class pixyz.distributions.**RelaxedBernoulli**(*temperature, cond_var=[], var=['x'], name='p', dim=None, **kwargs*)

Bases: pixyz.distributions.distributions.DistributionBase

distribution_name

log_likelihood(*x*)

Estimate the log likelihood of this distribution from inputs formatted by a dictionary.

x_dict [dict] Input samples.

log_like [torch.Tensor] Log-likelihood.

sample_mean(*x*)

1.2.4 FactorizedBernoulli

class pixyz.distributions.**FactorizedBernoulli**(*cond_var=[], var=['x'], name='p', dim=None, **kwargs*)

Bases: pixyz.distributions.exponential_distributions.Bernoulli

Generative Models of Visually Grounded Imagination

distribution_name

1.2.5 Categorical

class pixyz.distributions.**Categorical**(*cond_var=[], var=['x'], name='p', dim=None, **kwargs*)

Bases: pixyz.distributions.distributions.DistributionBase

distribution_name

sample_mean(*x*)

1.2.6 RelaxedCategorical

class pixyz.distributions.**RelaxedCategorical**(*temperature, cond_var=[], var=['x'], name='p', dim=None, **kwargs*)

Bases: pixyz.distributions.distributions.DistributionBase

distribution_name

log_likelihood(*x*)

Estimate the log likelihood of this distribution from inputs formatted by a dictionary.

x_dict [dict] Input samples.

log_like [torch.Tensor] Log-likelihood.

sample_mean (*x*)

1.3 Complex distributions

1.3.1 MixtureModel

class pixyz.distributions.**MixtureModel** (*distributions*, *prior*, *name='p'*)

Bases: *pixyz.distributions.distributions.Distribution*

Mixture models. $p(x) = \sum_i p(x|z=i)p(z=i)$

distributions [list] List of distributions.

prior [pixyz.Distribution.Categorical] Prior distribution of latent variable (i.e., the contribution rate). This should be a categorical distribution and the number of its category should be the same as the length of the distribution list.

```
>>> from pixyz.distributions import Normal, Categorical
>>> from pixyz.distributions.mixture_distributions import MixtureModel
>>>
>>> z_dim = 3  # the number of mixture
>>> x_dim = 2  # the input dimension.
>>>
>>> distributions = []  # the list of distributions
>>> for i in range(z_dim):
>>>     loc = torch.randn(x_dim)  # initialize the value of location (mean)
>>>     scale = torch.empty(x_dim).fill_(1.)  # initialize the value of scale␣
↪(variance)
>>>     distributions.append(Normal(loc=loc, scale=scale, var=["x"], name="p_%d"
↪%i))
>>>
>>> probs = torch.empty(z_dim).fill_(1. / z_dim)  # initialize the value of␣
↪probabilities
>>> prior = Categorical(probs=probs, var=["z"], name="prior")
>>>
>>> p = MixtureModel(distributions=distributions, prior=prior)
```

prob_text

prob_factorized_text

distribution_name

get_posterior_probs (*x_dict*)

sample (*batch_size=1*, *return_hidden=False*, ***kwargs*)

Sample variables of this distribution. If *cond_var* is not empty, we should set inputs as a dictionary format.

x [torch.Tensor, list, or dict] Input variables.

shape [tuple] Shape of samples. If set, *batch_size* and *dim* are ignored.

batch_size [int] Batch size of samples. This is set to 1 by default.

return_all [bool] Choose whether the output contains input variables.

reparam [bool] Choose whether we sample variables with reparameterized trick.

output [dict] Samples of this distribution.

log_likelihood_all_hidden (*x_dict*)
Estimate joint log-likelihood, log p(x, z), where input is *x*.

x_dict [dict] Input variables (including *var*).

loglike [torch.Tensor] dim=0 : the number of mixture dim=1 : the size of batch

log_likelihood (*x_dict*)
Estimate log-likelihood, log p(x).

x_dict [dict] Input variables (including *var*).

loglike [torch.Tensor] The log-likelihood value of x.

1.3.2 NormalPoE

class pixyz.distributions.**NormalPoE** (*prior*, *dists=[]*, ***kwargs*)
Bases: `torch.nn.modules.module.Module`

$p(z|x,y) \propto p(z)p(z|x)p(z|y)$

dists [list] Other distributions.

prior [Distribution] Prior distribution.

```
>>> poe = NormalPoE(c, [a, b])
```

get_params (*params*, ***kwargs*)

experts (*loc*, *scale*, *eps=1e-08*)

sample (*x=None*, *return_all=True*, ***kwargs*)

log_likelihood (*x*)

sample_mean (*x*, ***kwargs*)

1.4 Special distributions

1.4.1 Deterministic

class pixyz.distributions.**Deterministic** (***kwargs*)
Bases: `pixyz.distributions.distributions.Distribution`

Deterministic distribution (or degeneration distribution)

distribution_name

sample (*x={}*, *return_all=True*, ***kwargs*)
Sample variables of this distribution. If *cond_var* is not empty, we should set inputs as a dictionary format.

x [torch.Tensor, list, or dict] Input variables.

shape [tuple] Shape of samples. If set, *batch_size* and *dim* are ignored.

batch_size [int] Batch size of samples. This is set to 1 by default.

return_all [bool] Choose whether the output contains input variables.

reparam [bool] Choose whether we sample variables with reparameterized trick.

output [dict] Samples of this distribution.

1.4.2 DataDistribution

class pixyz.distributions.**DataDistribution**(*var*, *name='p_data'*)

Bases: *pixyz.distributions.distributions.Distribution*

Data distribution. TODO: Fix this behavior if multiplied with other distributions

distribution_name

sample(*x={}*, ***kwargs*)

Sample variables of this distribution. If *cond_var* is not empty, we should set inputs as a dictionary format.

x [torch.Tensor, list, or dict] Input variables.

shape [tuple] Shape of samples. If set, *batch_size* and *dim* are ignored.

batch_size [int] Batch size of samples. This is set to 1 by default.

return_all [bool] Choose whether the output contains input variables.

reparam [bool] Choose whether we sample variables with reparameterized trick.

output [dict] Samples of this distribution.

input_var

In DataDistribution, *input_var* is same as *var*.

1.4.3 CustomLikelihoodDistribution

class pixyz.distributions.**CustomLikelihoodDistribution**(*var=['x']*, *likelihood=None*, ***kwargs*)

Bases: *pixyz.distributions.distributions.Distribution*

input_var

In CustomLikelihoodDistribution, *input_var* is same as *var*.

distribution_name

log_likelihood(*x_dict*)

Estimate the log likelihood of this distribution from inputs formatted by a dictionary.

x_dict [dict] Input samples.

log_like [torch.Tensor] Log-likelihood.

1.5 Flow-based

1.5.1 PlanarFlow

class pixyz.distributions.**PlanarFlow**(*prior*, *dim*, *num_layers=1*, *var=[]*, ***kwargs*)
Bases: pixyz.distributions.flows.Flow

1.5.2 RealNVP

class pixyz.distributions.**RealNVP**(*prior*, *dim*, *num_multiscale_layers=2*, *var=[]*, *image=False*, *name='p'*, ***kwargs*)
Bases: *pixyz.distributions.distributions.Distribution*

prob_text

forward(*x*, *inverse=False*, *jacobian=False*)
When this class is inherited by DNNs, it is also intended that this method is overrided.

sample(*x={}*, *only_flow=False*, *return_all=True*, ***kwargs*)
Sample variables of this distribution. If *cond_var* is not empty, we should set inputs as a dictionary format.

x [torch.Tensor, list, or dict] Input variables.

shape [tuple] Shape of samples. If set, *batch_size* and *dim* are ignored.

batch_size [int] Batch size of samples. This is set to 1 by default.

return_all [bool] Choose whether the output contains input variables.

reparam [bool] Choose whether we sample variables with reparameterized trick.

output [dict] Samples of this distribution.

sample_inv(*x*, *return_all=True*, ***kwargs*)

log_likelihood(*x*)
Estimate the log likelihood of this distribution from inputs formatted by a dictionary.

x_dict [dict] Input samples.

log_like [torch.Tensor] Log-likelihood.

1.6 Operators

1.6.1 ReplaceVarDistribution

class pixyz.distributions.distributions.**ReplaceVarDistribution**(*a*, *replace_dict*)
Bases: *pixyz.distributions.distributions.Distribution*

Replace names of variables in Distribution.

a [pixyz.Distribution (not pixyz.MultiplyDistribution)] Distribution.

replace_dict [dict] Dictionary.

forward(**args*, ***kwargs*)
When this class is inherited by DNNs, it is also intended that this method is overrided.

get_params (*params_dict*)
This method aims to get parameters of this distributions from constant parameters set in initialization and outputs of DNNs.

params_dict [dict] Input parameters.

output_dict [dict] Output parameters

```
>>> print(dist_1.prob_text, dist_1.distribution_name)
p(x) Normal
>>> dist_1.get_params()
{"loc": 0, "scale": 1}
>>> print(dist_2.prob_text, dist_2.distribution_name)
p(x|z) Normal
>>> dist_1.get_params({"z": 1})
{"loc": 0, "scale": 1}
```

sample (*x={}*, *shape=None*, *batch_size=1*, *return_all=True*, *reparam=False*)
Sample variables of this distribution. If *cond_var* is not empty, we should set inputs as a dictionary format.

x [torch.Tensor, list, or dict] Input variables.

shape [tuple] Shape of samples. If set, *batch_size* and *dim* are ignored.

batch_size [int] Batch size of samples. This is set to 1 by default.

return_all [bool] Choose whether the output contains input variables.

reparam [bool] Choose whether we sample variables with reparameterized trick.

output [dict] Samples of this distribution.

log_likelihood (*x*)
Estimate the log likelihood of this distribution from inputs formatted by a dictionary.

x_dict [dict] Input samples.

log_like [torch.Tensor] Log-likelihood.

sample_mean (*x*)

input_var
Normally, *input_var* has same values as *cond_var*.

distribution_name

1.6.2 MarginalizeVarDistribution

class pixyz.distributions.distributions.**MarginalizeVarDistribution** (*a*, *marginalize_list*)

Bases: *pixyz.distributions.distributions.Distribution*

Marginalize variables in Distribution. $p(x) = \int p(x,z)dz$

a [pixyz.Distribution (not pixyz.DistributionBase)] Distribution.

marginalize_list [list] Variables to marginalize.

forward(**args*, ***kwargs*)
When this class is inherited by DNNs, it is also intended that this method is overrided.

get_params(*params_dict*)
This method aims to get parameters of this distributions from constant parameters set in initialization and outputs of DNNs.

params_dict [dict] Input parameters.

output_dict [dict] Output parameters

```
>>> print(dist_1.prob_text, dist_1.distribution_name)
p(x) Normal
>>> dist_1.get_params()
{"loc": 0, "scale": 1}
>>> print(dist_2.prob_text, dist_2.distribution_name)
p(x|z) Normal
>>> dist_1.get_params({"z": 1})
{"loc": 0, "scale": 1}
```

sample(*x={}*, *shape=None*, *batch_size=1*, *return_all=True*, *reparam=False*)
Sample variables of this distribution. If *cond_var* is not empty, we should set inputs as a dictionary format.

x [torch.Tensor, list, or dict] Input variables.

shape [tuple] Shape of samples. If set, *batch_size* and *dim* are ignored.

batch_size [int] Batch size of samples. This is set to 1 by default.

return_all [bool] Choose whether the output contains input variables.

reparam [bool] Choose whether we sample variables with reparameterized trick.

output [dict] Samples of this distribution.

log_likelihood(*x_dict*)
Estimate the log likelihood of this distribution from inputs formatted by a dictionary.

x_dict [dict] Input samples.

log_like [torch.Tensor] Log-likelihood.

sample_mean(*x*)

input_var
Normally, *input_var* has same values as *cond_var*.

distribution_name

prob_factorized_text

1.6.3 MultiplyDistribution

class pixyz.distributions.distributions.**MultiplyDistribution**(*a*, *b*)
Bases: *pixyz.distributions.distributions.Distribution*

Multiply by given distributions, e.g, $p(x, y|z) = p(x|z, y)p(y|z)$. In this class, it is checked if two distributions can be multiplied.

p(x|z)p(z|y) -> Valid

p(x|z)p(y|z) -> Valid

p(x|z)p(y|a) -> Valid

p(x|z)p(z|x) -> Invalid (recursive)

p(x|z)p(x|y) -> Invalid (conflict)

a [pixyz.Distribution] Distribution.

b [pixyz.Distribution] Distribution.

```
>>> p_multi = MultipleDistribution([a, b])
>>> p_multi = a * b
```

inh_var

input_var
Normally, *input_var* has same values as *cond_var*.

prob_factorized_text

sample(*x={}*, *shape=None*, *batch_size=1*, *return_all=True*, *reparam=False*)
Sample variables of this distribution. If *cond_var* is not empty, we should set inputs as a dictionary format.

x [torch.Tensor, list, or dict] Input variables.

shape [tuple] Shape of samples. If set, *batch_size* and *dim* are ignored.

batch_size [int] Batch size of samples. This is set to 1 by default.

return_all [bool] Choose whether the output contains input variables.

reparam [bool] Choose whether we sample variables with reparameterized trick.

output [dict] Samples of this distribution.

log_likelihood(*x*)
Estimate the log likelihood of this distribution from inputs formatted by a dictionary.

x_dict [dict] Input samples.

log_like [torch.Tensor] Log-likelihood.

1.7 Functions

pixyz.distributions.distributions.**sum_samples**(*samples*)

CHAPTER 2

pixyz.losses (Loss API)

2.1 Loss

class pixyz.losses.losses.**Loss**(*p1*, *p2=None*, *input_var=None*)

Bases: object

input_var

loss_text

mean()

sum()

estimate(*x={}*, ***kwargs*)

train(*x={}*, ***kwargs*)

Train the implicit (adversarial) loss function.

test(*x={}*, ***kwargs*)

Test the implicit (adversarial) loss function.

2.2 Negative expected value of log-likelihood (entropy)

2.2.1 CrossEntropy

class pixyz.losses.**CrossEntropy**(*p1*, *p2*, *input_var=None*)

Bases: *pixyz.losses.losses.Loss*

Cross entropy, a.k.a., the negative expected value of log-likelihood (Monte Carlo approximation).

$$-\mathbb{E}_{q(x)}[\log p(x)] \approx -\frac{1}{L}\sum_{l=1}^{L}\log p(x_l),$$

where $x_l \sim q(x)$.

loss_text

estimate(*x={}*)

2.2.2 Entropy

class pixyz.losses.**Entropy**(*p1*, *input_var=None*)
Bases: *pixyz.losses.losses.Loss*

Entropy (Monte Carlo approximation).

$$-\mathbb{E}_{p(x)}[\log p(x)] \approx -\frac{1}{L}\sum_{l=1}^{L}\log p(x_l),$$

where $x_l \sim p(x)$.

Note: This class is a special case of the *CrossEntropy* class. You can get the same result with *CrossEntropy*.

loss_text

estimate(*x={}*)

2.2.3 StochasticReconstructionLoss

class pixyz.losses.**StochasticReconstructionLoss**(*encoder*, *decoder*, *input_var=None*)
Bases: *pixyz.losses.losses.Loss*

Reconstruction Loss (Monte Carlo approximation).

$$-\mathbb{E}_{q(z|x)}[\log p(x|z)] \approx -\frac{1}{L}\sum_{l=1}^{L}\log p(x|z_l),$$

where $z_l \sim q(z|x)$.

Note: This class is a special case of the *CrossEntropy* class. You can get the same result with *CrossEntropy*.

loss_text

estimate(*x={}*)

2.3 Negative log-likelihood

2.3.1 NLL

class pixyz.losses.**NLL**(*p*, *input_var=None*)
Bases: *pixyz.losses.losses.Loss*

Negative log-likelihood.

$$\log p(x)$$

loss_text

estimate(*x={}*)

2.4 Lower bound

2.4.1 ELBO

class pixyz.losses.**ELBO**(*p*, *approximate_dist*, *input_var=None*)

Bases: `pixyz.losses.losses.Loss`

The evidence lower bound (Monte Carlo approximation).

$$\mathbb{E}_{q(z|x)}[\log \frac{p(x,z)}{q(z|x)}] \approx \frac{1}{L}\sum_{l=1}^{L} \log p(x, z_l),$$

where $z_l \sim q(z|x)$.

loss_text

estimate(*x={}*, *batch_size=None*)

2.5 Divergence

2.5.1 KullbackLeibler

class pixyz.losses.**KullbackLeibler**(*p1*, *p2*, *input_var=None*)

Bases: `pixyz.losses.losses.Loss`

Kullback-Leibler divergence (analytical).

$$D_{KL}[p||q] = \mathbb{E}_{p(x)}[\log \frac{p(x)}{q(x)}]$$

loss_text

estimate(*x*, ***kwargs*)

2.6 Similarity

2.6.1 SimilarityLoss

class pixyz.losses.**SimilarityLoss**(*p1, p2, input_var=None, var=['z'], margin=0*)

Bases: `pixyz.losses.losses.Loss`

Learning Modality-Invariant Representations for Speech and Images (Leidai et. al.)

estimate(*x*)

2.6.2 MultiModalContrastivenessLoss

class pixyz.losses.**MultiModalContrastivenessLoss**(*p1*, *p2*, *input_var=None*, *margin=0.5*)

Bases: `pixyz.losses.losses.Loss`

Disentangling by Partitioning: A Representation Learning Framework for Multimodal Sensory Data

estimate(*x*)

2.7 Adversarial loss (GAN loss)

2.7.1 AdversarialJensenShannon

class pixyz.losses.**AdversarialJensenShannon**(*p, q, discriminator, input_var=None, optimizer=<class 'torch.optim.adam.Adam'>, optimizer_params={}, inverse_g_loss=True*)

Bases: `pixyz.losses.adversarial_loss.AdversarialLoss`

Jensen-Shannon divergence (adversarial training).

$$D_{JS}[p(x)||q(x)] \leq 2 \cdot D_{JS}[p(x)||q(x)] + 2\log 2 = \mathbb{E}_{p(x)}[\log d^*(x)] + \mathbb{E}_{q(x)}[\log(1-d^*(x))],$$

where $d^*(x) = \arg\max_d \mathbb{E}_{p(x)}[\log d(x)] + \mathbb{E}_{q(x)}[\log(1-d(x))]$.

loss_text

estimate(*x={}, discriminator=False*)

d_loss(*y1, y2, batch_size*)

g_loss(*y1, y2, batch_size*)

2.7.2 AdversarialKullbackLeibler

class pixyz.losses.**AdversarialKullbackLeibler**(*q, p, discriminator, **kwargs*)

Bases: `pixyz.losses.adversarial_loss.AdversarialLoss`

Kullback-Leibler divergence (adversarial training).

$$D_{KL}[q(x)||p(x)] = \mathbb{E}_{q(x)}[\log \frac{q(x)}{p(x)}] = \mathbb{E}_{q(x)}[\log \frac{d^*(x)}{1-d^*(x)}],$$

where $d^*(x) = \arg\max_d \mathbb{E}_{p(x)}[\log d(x)] + \mathbb{E}_{q(x)}[\log(1-d(x))]$.

Note that this divergence is minimized to close q to p.

loss_text

estimate(*x={}, discriminator=False*)

g_loss(*y1, batch_size*)

d_loss(*y1, y2, batch_size*)

2.7.3 AdversarialWassersteinDistance

class pixyz.losses.**AdversarialWassersteinDistance**(*p, q, discriminator, clip_value=0.01, **kwargs*)

Bases: `pixyz.losses.adversarial_loss.AdversarialJensenShannon`

Wasserstein distance (adversarial training).

$$W(p,q) = \sup_{||d||_L \leq 1} \mathbb{E}_{p(x)}[d(x)] - \mathbb{E}_{q(x)}[d(x)]$$

loss_text

d_loss (*y1*, *y2*, **args*, ***kwargs*)

g_loss (*y1*, *y2*, **args*, ***kwargs*)

train (*train_x*, ***kwargs*)
Train the implicit (adversarial) loss function.

2.8 Loss for special purpose

2.8.1 Parameter

class pixyz.losses.losses.**Parameter** (*input_var*)
Bases: *pixyz.losses.losses.Loss*

estimate (*x={}*, ***kwargs*)

loss_text

2.9 Operators

2.9.1 LossOperator

class pixyz.losses.losses.**LossOperator** (*loss1*, *loss2*)
Bases: *pixyz.losses.losses.Loss*

loss_text

estimate (*x={}*, ***kwargs*)

train (*x*, ***kwargs*)
TODO: Fix

test (*x*, ***kwargs*)
TODO: Fix

2.9.2 LossSelfOperator

class pixyz.losses.losses.**LossSelfOperator** (*loss1*)
Bases: *pixyz.losses.losses.Loss*

train (*x={}*, ***kwargs*)
Train the implicit (adversarial) loss function.

test (*x={}*, ***kwargs*)
Test the implicit (adversarial) loss function.

2.9.3 AddLoss

class pixyz.losses.losses.**AddLoss** (*loss1*, *loss2*)
Bases: *pixyz.losses.losses.LossOperator*

loss_text

estimate (*x={}*, ***kwargs*)

2.9.4 SubLoss

class pixyz.losses.losses.**SubLoss**(*loss1*, *loss2*)
Bases: *pixyz.losses.losses.LossOperator*

loss_text

estimate(*x={}*, ***kwargs*)

2.9.5 MulLoss

class pixyz.losses.losses.**MulLoss**(*loss1*, *loss2*)
Bases: *pixyz.losses.losses.LossOperator*

loss_text

estimate(*x={}*, ***kwargs*)

2.9.6 DivLoss

class pixyz.losses.losses.**DivLoss**(*loss1*, *loss2*)
Bases: *pixyz.losses.losses.LossOperator*

loss_text

estimate(*x={}*, ***kwargs*)

2.9.7 NegLoss

class pixyz.losses.losses.**NegLoss**(*loss1*)
Bases: *pixyz.losses.losses.LossSelfOperator*

loss_text

estimate(*x={}*, ***kwargs*)

2.9.8 BatchMean

class pixyz.losses.losses.**BatchMean**(*loss1*)
Bases: *pixyz.losses.losses.LossSelfOperator*

Loss averaged over batch data.

$$\mathbb{E}_{p_{data}(x)}[\mathcal{L}(x)] \approx \frac{1}{N}\sum_{i=1}^N \mathcal{L}(x_i),$$

where $x_i \sim p_{data}(x)$ and $\mathcal{L}$ is a loss function.

loss_text

estimate(*x={}*, ***kwargs*)

2.9.9 BatchSum

class pixyz.losses.losses.**BatchSum**(*loss1*)

Bases: *pixyz.losses.losses.LossSelfOperator*

Loss summed over batch data.

$$\sum_{i=1}^{N} \mathcal{L}(x_i),$$

where $x_i \sim p_{data}(x)$ and $\mathcal{L}$ is a loss function.

loss_text

estimate(*x={}*, ***kwargs*)

CHAPTER 3

pixyz.models (Model API)

3.1 Model

class pixyz.models.**Model**(*loss*, *test_loss=None*, *distributions=[]*, *optimizer=<class 'torch.optim.adam.Adam'>*, *optimizer_params={}*)

Bases: `object`

set_loss(*loss*, *test_loss=None*)

train(*train_x={}*, ***kwargs*)

test(*test_x={}*, ***kwargs*)

3.2 Pre-implementation models

3.2.1 ML

class pixyz.models.**ML**(*p*, *other_distributions=[]*, *optimizer=<class 'torch.optim.adam.Adam'>*, *optimizer_params={}*)

Bases: `pixyz.models.model.Model`

Maximum Likelihood (log-likelihood)

train(*train_x={}*, ***kwargs*)

test(*test_x={}*, ***kwargs*)

3.2.2 VAE

class pixyz.models.**VAE**(*encoder*, *decoder*, *other_distributions=[]*, *regularizer=[]*, *optimizer=<class 'torch.optim.adam.Adam'>*, *optimizer_params={}*)

Bases: `pixyz.models.model.Model`

Variational Autoencoder

[Kingma+ 2013] Auto-Encoding Variational Bayes

train(*train_x={}*, ***kwargs*)

test(*test_x={}*, ***kwargs*)

3.2.3 VI

class pixyz.models.**VI**(*p*, *approximate_dist*, *other_distributions=[]*, *optimizer=<class 'torch.optim.adam.Adam'>*, *optimizer_params={}*)

Bases: `pixyz.models.model.Model`

Variational Inference (Amortized inference)

train(*train_x={}*, ***kwargs*)

test(*test_x={}*, ***kwargs*)

3.2.4 GAN

class pixyz.models.**GAN**(*p_data*, *p*, *discriminator*, *optimizer=<class 'torch.optim.adam.Adam'>*, *optimizer_params={}*, *d_optimizer=<class 'torch.optim.adam.Adam'>*, *d_optimizer_params={}*)

Bases: `pixyz.models.model.Model`

Generative Adversarial Network

train(*train_x={}*, *adversarial_loss=True*, ***kwargs*)

test(*test_x={}*, *adversarial_loss=True*, ***kwargs*)

CHAPTER 4

pixyz.utils

pixyz.utils.**set_epsilon**(*eps*)

pixyz.utils.**epsilon**()

pixyz.utils.**get_dict_values**(*dicts*, *keys*, *return_dict=False*)

pixyz.utils.**delete_dict_values**(*dicts*, *keys*)

pixyz.utils.**detach_dict**(*dicts*)

pixyz.utils.**replace_dict_keys**(*dicts*, *replace_list_dict*)

pixyz.utils.**tolist**(*a*)

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index

A

B

C

D

E

F

G

I

K

L

V