

---

# Pixyz Documentation

**masa**

**Apr 18, 2019**



---

## Package Reference

---

<b>1</b>	<b>pixyz.distributions (Distribution API)</b>	<b>3</b>
<b>2</b>	<b>pixyz.losses (Loss API)</b>	<b>13</b>
<b>3</b>	<b>pixyz.models (Model API)</b>	<b>21</b>
<b>4</b>	<b>pixyz.utils</b>	<b>23</b>
<b>5</b>	<b>Indices and tables</b>	<b>25</b>
	<b>Python Module Index</b>	<b>27</b>



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



---

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.

**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.

**get\_log\_prob** (*\*args*, *\*\*kwargs*)

**log\_prob** (*sum\_features=True*, *feature\_dims=None*)

**prob** (*sum\_features=True*, *feature\_dims=None*)

**log\_likelihood** (*\*args*, *\*\*kwargs*)

**forward** (*\*args*, *\*\*kwargs*)

When this class is inherited by DNNs, it is also intended that this method is overridden.

**sample\_mean** (*x*)

**sample\_variance** (*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

Normal distribution parameterized by *loc* and *scale*.

**distribution\_name**



### 1.2.2 Laplace

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

    Laplace distribution parameterized by loc and scale.

    distribution_name
```

### 1.2.3 Bernoulli

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

    Bernoulli distribution parameterized by probs.

    distribution_name
```

### 1.2.4 RelaxedBernoulli

```
class pixyz.distributions.RelaxedBernoulli (temperature=tensor(0.1000), cond_var=[],
                                             var=['x'], name='p', dim=None, **kwargs)
    Bases: pixyz.distributions.distributions.DistributionBase

    Relaxed (reparameterizable) Bernoulli distribution parameterized by probs.

    distribution_name

    set_distribution (x={}, sampling=True, **kwargs)
        Require self.params_keys and self.DistributionTorch

        x : dict

        sampling : bool
```

### 1.2.5 FactorizedBernoulli

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

    Factorized Bernoulli distribution parameterized by probs.

    See Generative Models of Visually Grounded Imagination

    distribution_name

    get_log_prob (x)
        x_dict : dict

        sum_features : bool

        feature_dims : None or list

        log_prob : torch.Tensor
```

### 1.2.6 Categorical

```
class pixyz.distributions.Categorical (cond_var=[], var=['x'], name='p', dim=None,
                                     **kwargs)
    Bases: pixyz.distributions.distributions.DistributionBase
    Categorical distribution parameterized by probs.
    distribution_name
```

### 1.2.7 RelaxedCategorical

```
class pixyz.distributions.RelaxedCategorical (temperature=tensor(0.1000), cond_var=[],
                                              var=['x'], name='p', dim=None,
                                              **kwargs)
    Bases: pixyz.distributions.distributions.DistributionBase
    Relaxed (reparameterizable) categorical distribution parameterized by probs.
    distribution_name
    set_distribution (x={}, sampling=True, **kwargs)
        Require self.params_keys and self.DistributionTorch
        x : dict
        sampling : bool
    sample_mean (x={})
    sample_variance (x={})
```

### 1.2.8 Beta

```
class pixyz.distributions.Beta (cond_var=[], var=['x'], name='p', dim=None, **kwargs)
    Bases: pixyz.distributions.distributions.DistributionBase
    Beta distribution parameterized by concentration1 and concentration0.
    distribution_name
```

### 1.2.9 Dirichlet

```
class pixyz.distributions.Dirichlet (cond_var=[], var=['x'], name='p', dim=None,
                                     **kwargs)
    Bases: pixyz.distributions.distributions.DistributionBase
    Dirichlet distribution parameterized by concentration.
    distribution_name
```

### 1.2.10 Gamma

```
class pixyz.distributions.Gamma (cond_var=[], var=['x'], name='p', dim=None, **kwargs)
    Bases: pixyz.distributions.distributions.DistributionBase
    Gamma distribution parameterized by concentration and rate.
    distribution_name
```

## 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**

**posterior** (*name=None*)

**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.

**get\_log\_prob** (*x\_dict, return\_hidden=False, \*\*kwargs*)

Evaluate log-pdf, log  $p(x)$  (if *return\_hidden=False*) or log  $p(x, z)$  (if *return\_hidden=True*).

**x\_dict** [dict] Input variables (including *var*).

**return\_hidden** : bool (False as default)

**log\_prob** [torch.Tensor] The log-pdf value of *x*.

**return\_hidden = 0** : dim=0 : the size of batch

**return\_hidden = 1** : dim=0 : the number of mixture dim=1 : the size of batch

### 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])
```

**set\_distribution** (*x*={}, *\*\*kwargs*)

**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.

**sample\_mean** (*x*)

## 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.

**sample\_mean** (*x*)

**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*)

## 1.5 Flow-based

### 1.5.1 PlanarFlow

### 1.5.2 RealNVP

## 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 overridden.

**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.

**get\_log\_prob** (x\_dict, \*\*kwargs)

*x\_dict*: dict

torch.Tensor

In

**sample\_mean** ( $x$ )

**sample\_variance** ( $x$ )

**input\_var**  
Normally, *input\_var* has same values as *cond\_var*.

**distribution\_name**

## 1.6.2 MarginalizeVarDistribution

**class** pixyz.distributions.distributions.**MarginalizeVarDistribution** ( $a$ ,  
*marginal-  
ize\_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 overridden.

**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.

**sample\_mean** ( $x$ )

**sample\_variance** ( $x$ )

**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.

**get\_log\_prob** (*x*, *sum\_features=True*, *feature\_dims=None*)

## 1.7 Functions

*pixyz.distributions.distributions.sum\_samples* (*samples*)



## 2.1 Loss

```
class pixyz.losses.losses.Loss (p, q=None, input_var=None)
    Bases: object

    input_var
    loss_text
    abs ()
    mean ()
    sum ()
    eval (x={}, return_dict=False, **kwargs)
    expectation (p, input_var=None)
    estimate (*args, **kwargs)
```

## 2.2 Negative expected value of log-likelihood (entropy)

### 2.2.1 CrossEntropy

```
class pixyz.losses.CrossEntropy (p, q, input_var=None)
    Bases: pixyz.losses.losses.SetLoss

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

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

where  $x_l \sim p(x)$ .

**Note:** This class is a special case of the *Expectation* class.

## 2.2.2 Entropy

**class** pixyz.losses.**Entropy** (*p*, *input\_var=None*)

Bases: pixyz.losses.losses.SetLoss

Entropy (Monte Carlo approximation).

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

where  $x_l \sim p(x)$ .

**Note:** This class is a special case of the *Expectation* class.

## 2.2.3 StochasticReconstructionLoss

**class** pixyz.losses.**StochasticReconstructionLoss** (*encoder*, *decoder*, *input\_var=None*)

Bases: pixyz.losses.losses.SetLoss

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 *Expectation* class.

## 2.2.4 LossExpectation

## 2.3 Negative log-likelihood

### 2.3.1 NLL

## 2.4 Lower bound

### 2.4.1 ELBO

**class** pixyz.losses.**ELBO** (*p*, *q*, *input\_var=None*)

Bases: pixyz.losses.losses.SetLoss

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

**Note:** This class is a special case of the *Expectation* class.

## 2.5 Statistical distance

### 2.5.1 KullbackLeibler

**class** pixyz.losses.**KullbackLeibler** (*p, q, input\_var=None, dim=None*)

Bases: `pixyz.losses.losses.Loss`

Kullback-Leibler divergence (analytical).

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

**TODO:** This class seems to be slightly slower than this previous implementation (perhaps because of `set_distribution`).

**loss\_text**

### 2.5.2 WassersteinDistance

**class** pixyz.losses.**WassersteinDistance** (*p, q, metric=PairwiseDistance(), input\_var=None*)

Bases: `pixyz.losses.losses.Loss`

Wasserstein distance.

$$W(p, q) = \inf_{\Gamma \in \mathcal{P}(x_p \sim p, x_q \sim q)} \mathbb{E}_{(x_p, x_q) \sim \Gamma} [d(x_p, x_q)]$$

However, instead of the above true distance, this class computes the following one.

$$W'(p, q) = \mathbb{E}_{x_p \sim p, x_q \sim q} [d(x_p, x_q)].$$

Here,  $W'$  is the upper of  $W$  (i.e.,  $W \leq W'$ ), and these are equal when both  $p$  and  $q$  are degenerate (deterministic) distributions.

**loss\_text**

### 2.5.3 MMD

**class** pixyz.losses.**MMD** (*p, q, input\_var=None, kernel='gaussian', \*\*kernel\_params*)

Bases: `pixyz.losses.losses.Loss`

The Maximum Mean Discrepancy (MMD).

$$D_{MMD^2}[p||q] = \mathbb{E}_{p(x), p(x')} [k(x, x')] + \mathbb{E}_{q(x), q(x')} [k(x, x')] - 2\mathbb{E}_{p(x), q(x')} [k(x, x')]$$

where  $k(x, x')$  is any positive definite kernel.

**loss\_text**

## 2.6 Adversarial statistical distance (GAN loss)

### 2.6.1 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**

**d\_loss** (*y\_p, y\_q, batch\_size*)

**g\_loss** (*y\_p, y\_q, batch\_size*)

## 2.6.2 AdversarialKullbackLeibler

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

Bases: pixyz.losses.adversarial\_loss.AdversarialLoss

Kullback-Leibler divergence (adversarial training).

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

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

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

**loss\_text**

**g\_loss** (*y\_p, batch\_size*)

**d\_loss** (*y\_p, y\_q, batch\_size*)

## 2.6.3 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** (*y\_p, y\_q, \*args, \*\*kwargs*)

**g\_loss** (*y\_p, y\_q, \*args, \*\*kwargs*)

**train** (*train\_x, \*\*kwargs*)

## 2.7 Loss for sequential distributions

### 2.7.1 IterativeLoss

**class** pixyz.losses.**IterativeLoss** (*step\_loss, max\_iter=1, input\_var=None, series\_var=None, update\_value={}, slice\_step=None, timestep\_var='t'*)

Bases: pixyz.losses.losses.Loss

Iterative loss.

This class allows implementing an arbitrary model which requires iteration (e.g., auto-regressive models).

$$\mathcal{L} = \sum_{t=1}^T \mathcal{L}_{step}(x_t, h_t), \text{ where } x_t = f_{slice\_step}(x, t)$$

```
loss_text
slice_step_fn(t, x)
```

## 2.8 Loss for special purpose

### 2.8.1 Parameter

```
class pixyz.losses.losses.Parameter(input_var)
    Bases: pixyz.losses.losses.Loss
    loss_text
```

## 2.9 Operators

### 2.9.1 LossOperator

```
class pixyz.losses.losses.LossOperator(loss1, loss2)
    Bases: pixyz.losses.losses.Loss
    loss_text
    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)
    test(x={}, **kwargs)
```

### 2.9.3 AddLoss

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

## 2.9.4 SubLoss

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

    loss_text
```

## 2.9.5 MulLoss

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

    loss_text
```

## 2.9.6 DivLoss

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

    loss_text
```

## 2.9.7 NegLoss

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

    loss_text
```

## 2.9.8 AbsLoss

```
class pixyz.losses.losses.AbsLoss (loss1)
    Bases: pixyz.losses.losses.LossSelfOperator

    loss_text
```

## 2.9.9 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
```

### 2.9.10 BatchSum

**class** `pixyz.losses.losses.BatchSum(lossl)`  
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**





### 3.1 Model

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

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'>, opti-
                        mizer_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`



### p

- `pixyz.distributions`, 3
- `pixyz.losses`, 13
- `pixyz.models`, 21
- `pixyz.utils`, 23





## A

abs() (*pixyz.losses.losses.Loss method*), 13  
AbsLoss (*class in pixyz.losses.losses*), 18  
AddLoss (*class in pixyz.losses.losses*), 17  
AdversarialJensenShannon (*class in pixyz.losses*), 15  
AdversarialKullbackLeibler (*class in pixyz.losses*), 16  
AdversarialWassersteinDistance (*class in pixyz.losses*), 16

## B

BatchMean (*class in pixyz.losses.losses*), 18  
BatchSum (*class in pixyz.losses.losses*), 19  
Bernoulli (*class in pixyz.distributions*), 5  
Beta (*class in pixyz.distributions*), 6

## C

Categorical (*class in pixyz.distributions*), 6  
cond\_var (*pixyz.distributions.distributions.Distribution attribute*), 3  
CrossEntropy (*class in pixyz.losses*), 13  
CustomLikelihoodDistribution (*class in pixyz.distributions*), 9

## D

d\_loss() (*pixyz.losses.AdversarialJensenShannon method*), 16  
d\_loss() (*pixyz.losses.AdversarialKullbackLeibler method*), 16  
d\_loss() (*pixyz.losses.AdversarialWassersteinDistance method*), 16  
DataDistribution (*class in pixyz.distributions*), 9  
delete\_dict\_values() (*in module pixyz.utils*), 23  
detach\_dict() (*in module pixyz.utils*), 23  
Deterministic (*class in pixyz.distributions*), 8  
Dirichlet (*class in pixyz.distributions*), 6  
Distribution (*class in pixyz.distributions.distributions*), 3

distribution\_name (*pixyz.distributions.Bernoulli attribute*), 5  
distribution\_name (*pixyz.distributions.Beta attribute*), 6  
distribution\_name (*pixyz.distributions.Categorical attribute*), 6  
distribution\_name (*pixyz.distributions.CustomLikelihoodDistribution attribute*), 9  
distribution\_name (*pixyz.distributions.DataDistribution attribute*), 9  
distribution\_name (*pixyz.distributions.Deterministic attribute*), 8  
distribution\_name (*pixyz.distributions.Dirichlet attribute*), 6  
distribution\_name (*pixyz.distributions.distributions.Distribution attribute*), 3  
distribution\_name (*pixyz.distributions.distributions.MarginalizeVarDistribution attribute*), 12  
distribution\_name (*pixyz.distributions.distributions.ReplaceVarDistribution attribute*), 11  
distribution\_name (*pixyz.distributions.FactorizedBernoulli attribute*), 5  
distribution\_name (*pixyz.distributions.Gamma attribute*), 6  
distribution\_name (*pixyz.distributions.Laplace attribute*), 5  
distribution\_name (*pixyz.distributions.MixtureModel attribute*), 7  
distribution\_name (*pixyz.distributions.Normal attribute*), 4  
distribution\_name (*pixyz.distributions.RelaxedBernoulli attribute*), 5

distribution\_name  
(*pixyz.distributions.RelaxedCategorical*  
*tribute*), 6  
DivLoss (*class in pixyz.losses.losses*), 18

## E

ELBO (*class in pixyz.losses*), 14  
Entropy (*class in pixyz.losses*), 14  
epsilon() (*in module pixyz.utils*), 23  
estimate() (*pixyz.losses.losses.Loss method*), 13  
eval() (*pixyz.losses.losses.Loss method*), 13  
expectation() (*pixyz.losses.losses.Loss method*), 13  
experts() (*pixyz.distributions.NormalPoE method*), 8

## F

FactorizedBernoulli (*class in*  
*pixyz.distributions*), 5  
forward() (*pixyz.distributions.distributions.Distribution*  
*method*), 4  
forward() (*pixyz.distributions.distributions.MarginalizeVarDistribution*  
*method*), 11  
forward() (*pixyz.distributions.distributions.ReplaceVarDistribution*  
*method*), 10

## G

g\_loss() (*pixyz.losses.AdversarialJensenShannon*  
*method*), 16  
g\_loss() (*pixyz.losses.AdversarialKullbackLeibler*  
*method*), 16  
g\_loss() (*pixyz.losses.AdversarialWassersteinDistance*  
*method*), 16  
Gamma (*class in pixyz.distributions*), 6  
GAN (*class in pixyz.models*), 22  
get\_dict\_values() (*in module pixyz.utils*), 23  
get\_log\_prob() (*pixyz.distributions.distributions.Distribution*  
*method*), 4  
get\_log\_prob() (*pixyz.distributions.distributions.MultiplyDistribution*  
*method*), 12  
get\_log\_prob() (*pixyz.distributions.distributions.ReplaceVarDistribution*  
*method*), 10  
get\_log\_prob() (*pixyz.distributions.FactorizedBernoulli*  
*method*), 5  
get\_log\_prob() (*pixyz.distributions.MixtureModel*  
*method*), 7  
get\_params() (*pixyz.distributions.distributions.Distribution*  
*method*), 3  
get\_params() (*pixyz.distributions.distributions.MarginalizeVarDistribution*  
*method*), 11  
get\_params() (*pixyz.distributions.distributions.ReplaceVarDistribution*  
*method*), 10  
get\_params() (*pixyz.distributions.NormalPoE*  
*method*), 8

## I

inh\_var(*pixyz.distributions.distributions.MultiplyDistribution*  
*attribute*), 12  
input\_var(*pixyz.distributions.CustomLikelihoodDistribution*  
*attribute*), 9  
input\_var(*pixyz.distributions.DataDistribution* *at-*  
*tribute*), 9  
input\_var(*pixyz.distributions.distributions.Distribution*  
*attribute*), 3  
input\_var(*pixyz.distributions.distributions.MarginalizeVarDistribution*  
*attribute*), 11  
input\_var(*pixyz.distributions.distributions.MultiplyDistribution*  
*attribute*), 12  
input\_var(*pixyz.distributions.distributions.ReplaceVarDistribution*  
*attribute*), 11  
input\_var(*pixyz.losses.losses.Loss attribute*), 13  
IterativeLoss (*class in pixyz.losses*), 16

## K

KullbackLeibler (*class in pixyz.losses*), 15

## L

Laplace (*class in pixyz.distributions*), 5  
log\_likelihood() (*pixyz.distributions.CustomLikelihoodDistribution*  
*method*), 9  
log\_likelihood() (*pixyz.distributions.distributions.Distribution*  
*method*), 4  
log\_likelihood() (*pixyz.distributions.NormalPoE*  
*method*), 8  
log\_prob() (*pixyz.distributions.distributions.Distribution*  
*method*), 4  
Loss (*class in pixyz.losses.losses*), 13  
loss\_text (*pixyz.losses.AdversarialJensenShannon*  
*attribute*), 16  
loss\_text (*pixyz.losses.AdversarialKullbackLeibler*  
*attribute*), 16  
loss\_text (*pixyz.losses.AdversarialWassersteinDistance*  
*attribute*), 16  
loss\_text (*pixyz.losses.IterativeLoss attribute*), 17  
loss\_text (*pixyz.losses.KullbackLeibler attribute*), 15  
loss\_text (*pixyz.losses.losses.AbsLoss attribute*), 18  
loss\_text (*pixyz.losses.losses.AddLoss attribute*), 17  
loss\_text (*pixyz.losses.losses.BatchMean attribute*),  
18  
loss\_text (*pixyz.losses.losses.BatchSum attribute*), 19  
loss\_text (*pixyz.losses.losses.DivLoss attribute*), 18  
loss\_text (*pixyz.losses.losses.Loss attribute*), 13  
loss\_text (*pixyz.losses.losses.LossOperator* *at-*  
*tribute*), 17  
loss\_text (*pixyz.losses.losses.MulLoss attribute*), 18  
loss\_text (*pixyz.losses.losses.NegLoss attribute*), 18  
loss\_text (*pixyz.losses.losses.Parameter attribute*),  
17

loss\_text (*pixyz.losses.losses.SubLoss attribute*), 18  
 loss\_text (*pixyz.losses.MMD attribute*), 15  
 loss\_text (*pixyz.losses.WassersteinDistance attribute*), 15  
 LossOperator (*class in pixyz.losses.losses*), 17  
 LossSelfOperator (*class in pixyz.losses.losses*), 17

## M

marginalize\_var() (*pixyz.distributions.distributions.Distribution method*), 4  
 MarginalizeVarDistribution (*class in pixyz.distributions.distributions*), 11  
 mean() (*pixyz.losses.losses.Loss method*), 13  
 MixtureModel (*class in pixyz.distributions*), 7  
 ML (*class in pixyz.models*), 21  
 MMD (*class in pixyz.losses*), 15  
 Model (*class in pixyz.models*), 21  
 MulLoss (*class in pixyz.losses.losses*), 18  
 MultiplyDistribution (*class in pixyz.distributions.distributions*), 12

## N

name (*pixyz.distributions.distributions.Distribution attribute*), 3  
 NegLoss (*class in pixyz.losses.losses*), 18  
 Normal (*class in pixyz.distributions*), 4  
 NormalPoE (*class in pixyz.distributions*), 8

## P

Parameter (*class in pixyz.losses.losses*), 17  
 pixyz.distributions (*module*), 3  
 pixyz.losses (*module*), 13  
 pixyz.models (*module*), 21  
 pixyz.utils (*module*), 23  
 posterior() (*pixyz.distributions.MixtureModel method*), 7  
 prob() (*pixyz.distributions.distributions.Distribution method*), 4  
 prob\_factorized\_text (*pixyz.distributions.distributions.Distribution attribute*), 3  
 prob\_factorized\_text (*pixyz.distributions.distributions.MarginalizeVarDistribution attribute*), 12  
 prob\_factorized\_text (*pixyz.distributions.distributions.MultiplyDistribution attribute*), 12  
 prob\_factorized\_text (*pixyz.distributions.MixtureModel attribute*), 7  
 prob\_text (*pixyz.distributions.distributions.Distribution attribute*), 3  
 prob\_text (*pixyz.distributions.MixtureModel attribute*), 7

## R

RelaxedBernoulli (*class in pixyz.distributions*), 5  
 RelaxedCategorical (*class in pixyz.distributions*), 6  
 replace\_dict\_keys() (*in module pixyz.utils*), 23  
 replace\_var() (*pixyz.distributions.distributions.Distribution method*), 4  
 ReplaceVarDistribution (*class in pixyz.distributions.distributions*), 10

## S

sample() (*pixyz.distributions.DataDistribution method*), 9  
 sample() (*pixyz.distributions.Deterministic method*), 8  
 sample() (*pixyz.distributions.distributions.Distribution method*), 4  
 sample() (*pixyz.distributions.distributions.MarginalizeVarDistribution method*), 11  
 sample() (*pixyz.distributions.distributions.MultiplyDistribution method*), 12  
 sample() (*pixyz.distributions.distributions.ReplaceVarDistribution method*), 10  
 sample() (*pixyz.distributions.MixtureModel method*), 7  
 sample() (*pixyz.distributions.NormalPoE method*), 8  
 sample\_mean() (*pixyz.distributions.DataDistribution method*), 9  
 sample\_mean() (*pixyz.distributions.Deterministic method*), 8  
 sample\_mean() (*pixyz.distributions.distributions.Distribution method*), 4  
 sample\_mean() (*pixyz.distributions.distributions.MarginalizeVarDistribution method*), 11  
 sample\_mean() (*pixyz.distributions.distributions.ReplaceVarDistribution method*), 11  
 sample\_mean() (*pixyz.distributions.NormalPoE method*), 8  
 sample\_mean() (*pixyz.distributions.RelaxedCategorical method*), 6  
 sample\_variance() (*pixyz.distributions.distributions.Distribution method*), 4  
 sample\_variance() (*pixyz.distributions.distributions.MarginalizeVarDistribution method*), 11  
 sample\_variance() (*pixyz.distributions.distributions.ReplaceVarDistribution method*), 11  
 sample\_variance() (*pixyz.distributions.RelaxedCategorical method*), 6  
 set\_distribution() (*pixyz.distributions.NormalPoE method*), 8

`set_distribution()`  
    (*pixyz.distributions.RelaxedBernoulli method*),  
    5  
`set_distribution()`  
    (*pixyz.distributions.RelaxedCategorical*  
    *method*), 6  
`set_epsilon()` (*in module pixyz.utils*), 23  
`set_loss()` (*pixyz.models.Model method*), 21  
`slice_step_fn()` (*pixyz.losses.IterativeLoss*  
    *method*), 17  
`StochasticReconstructionLoss` (*class in*  
    *pixyz.losses*), 14  
`SubLoss` (*class in pixyz.losses.losses*), 18  
`sum()` (*pixyz.losses.losses.Loss method*), 13  
`sum_samples()` (*in module*  
    *pixyz.distributions.distributions*), 12

## T

`test()` (*pixyz.losses.losses.LossOperator method*), 17  
`test()` (*pixyz.losses.losses.LossSelfOperator method*),  
    17  
`test()` (*pixyz.models.GAN method*), 22  
`test()` (*pixyz.models.ML method*), 21  
`test()` (*pixyz.models.Model method*), 21  
`test()` (*pixyz.models.VAE method*), 22  
`test()` (*pixyz.models.VI method*), 22  
`tolist()` (*in module pixyz.utils*), 23  
`train()` (*pixyz.losses.AdversarialWassersteinDistance*  
    *method*), 16  
`train()` (*pixyz.losses.losses.LossOperator method*), 17  
`train()` (*pixyz.losses.losses.LossSelfOperator*  
    *method*), 17  
`train()` (*pixyz.models.GAN method*), 22  
`train()` (*pixyz.models.ML method*), 21  
`train()` (*pixyz.models.Model method*), 21  
`train()` (*pixyz.models.VAE method*), 22  
`train()` (*pixyz.models.VI method*), 22

## V

`VAE` (*class in pixyz.models*), 21  
`var` (*pixyz.distributions.distributions.Distribution*  
    *attribute*), 3  
`VI` (*class in pixyz.models*), 22

## W

`WassersteinDistance` (*class in pixyz.losses*), 15