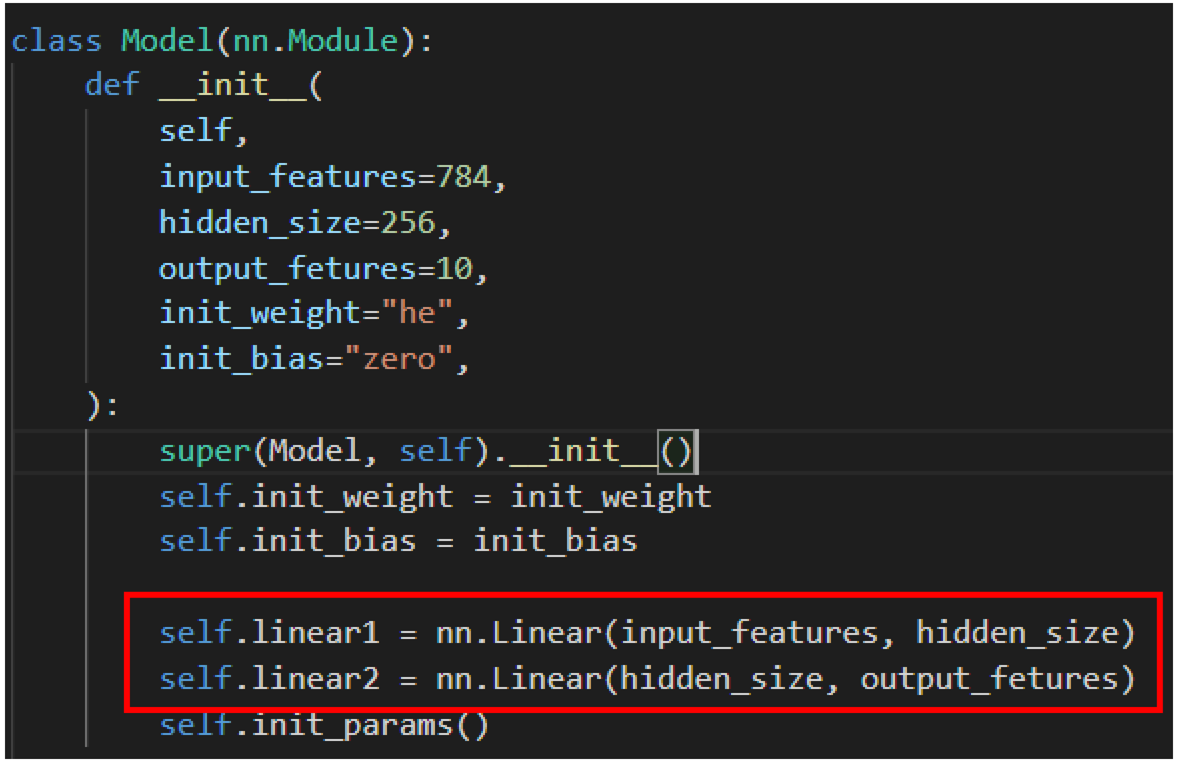
Day 11 Presentation

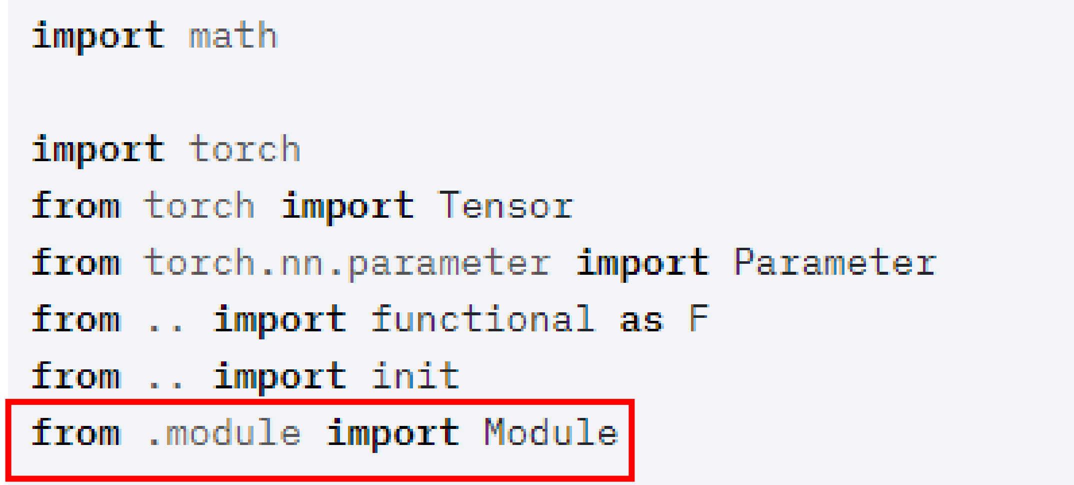
torch.nn.Module

torch.nn.Module

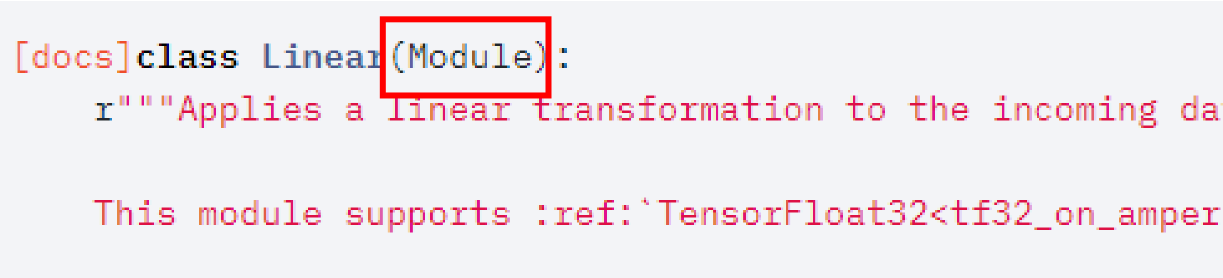
- 모든 neural network의 Base class
- 직접 model을 만들 때도 이 class의 subclass여야함
- Module 내부에는 다른 Module이 포함되어 있을 수 있음.



```python
class Model(nn.Module):
    def __init__(
        self,
        input_features=784,
        hidden_size=256,
        output_fetures=10,
        init_weight="he",
        init_bias="zero",
    ):
        super(Model, self).__init__()
        self.init_weight = init_weight
        self.init_bias = init_bias

        self.linear1 = nn.Linear(input_features, hidden_size)
        self.linear2 = nn.Linear(hidden_size, output_fetures)
        self.init_params()
```

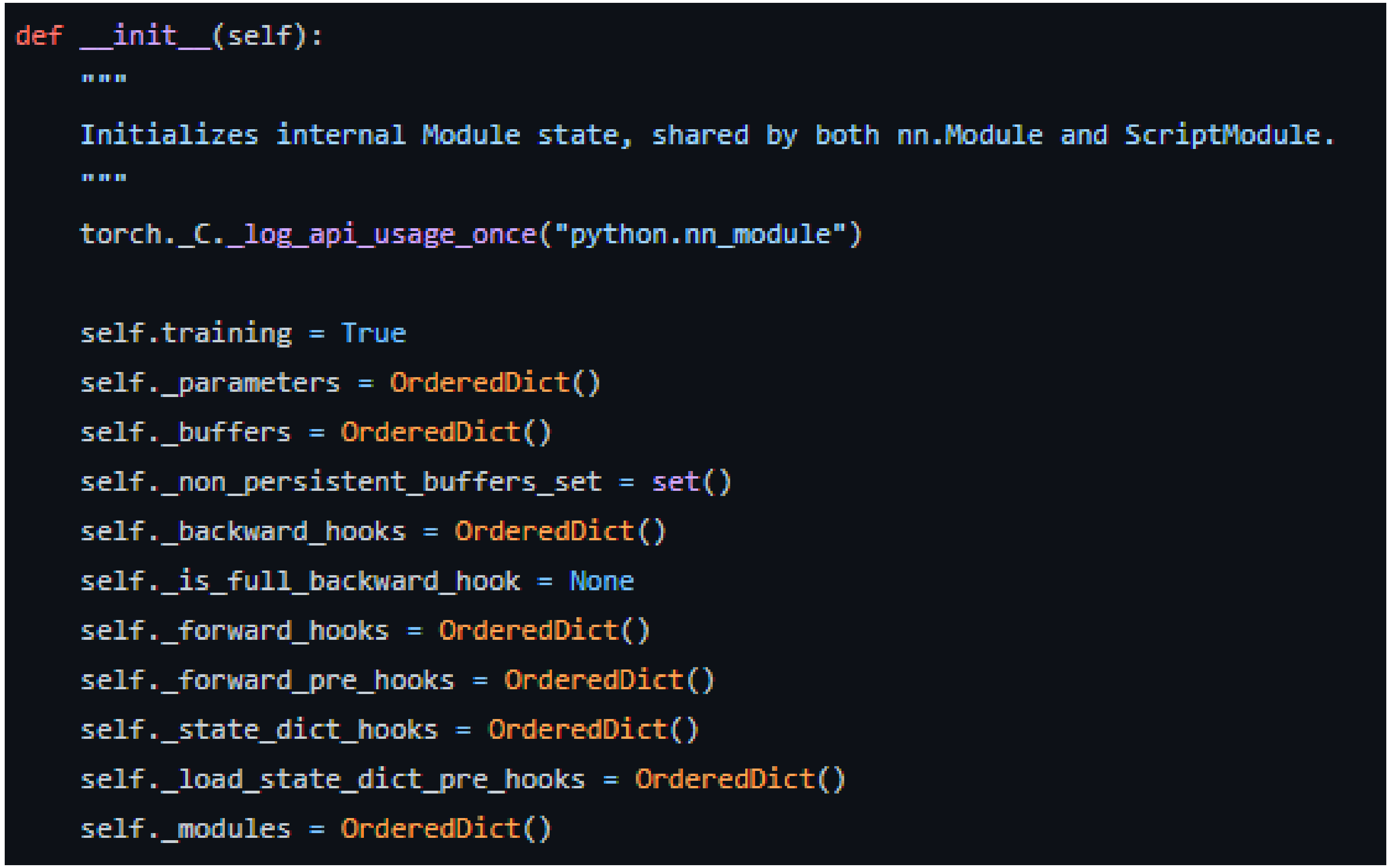


```python
import math

import torch
from torch import Tensor
from torch.nn.parameter import Parameter
from .. import functional as F
from .. import init
from .module import Module
```



```python
[docs]class Linear(Module):
    r"""Applies a linear transformation to the incoming da

    This module supports :ref:`TensorFloat32<tf32_on_amper
```

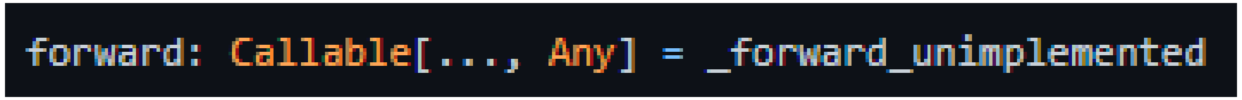
__init__(self)



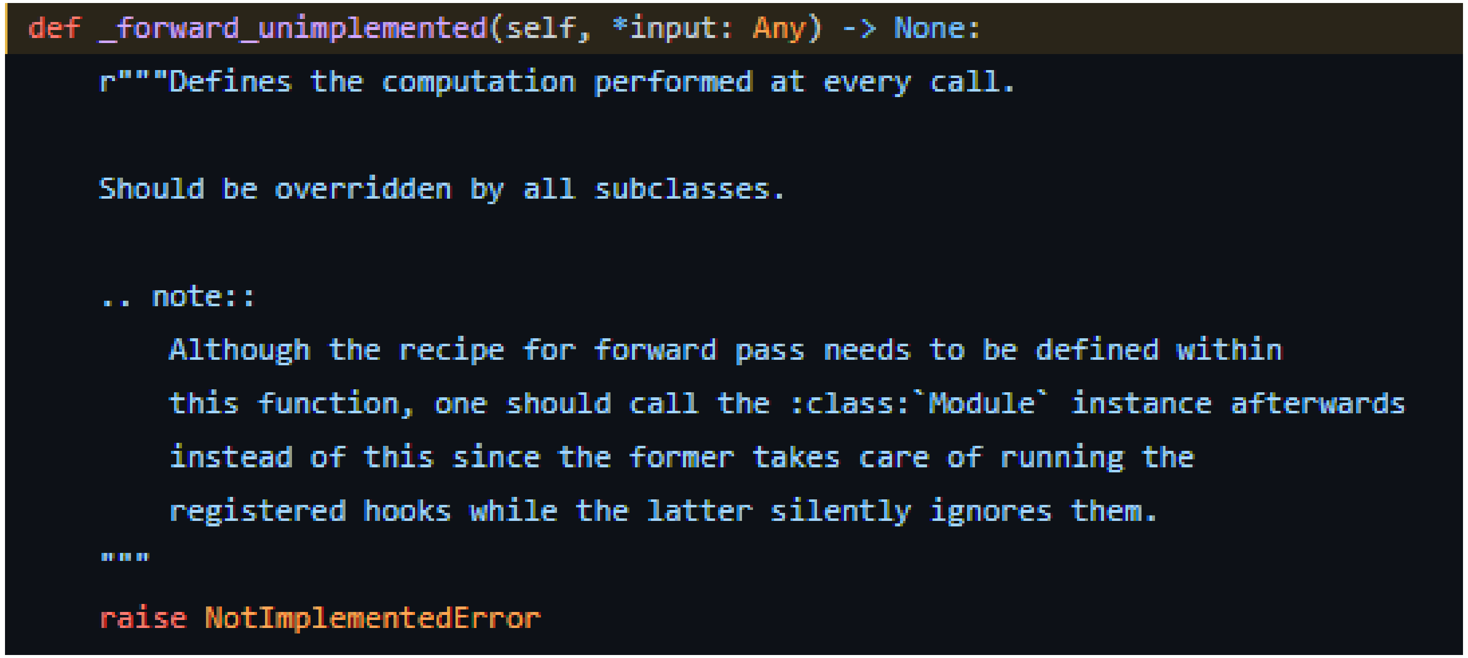
```
def __init__(self):
    """
    Initializes internal Module state, shared by both nn.Module and ScriptModule.
    """
    torch._C._log_api_usage_once("python.nn_module")

    self.training = True
    self._parameters = OrderedDict()
    self._buffers = OrderedDict()
    self._non_persistent_buffers_set = set()
    self._backward_hooks = OrderedDict()
    self._is_full_backward_hook = None
    self._forward_hooks = OrderedDict()
    self._forward_pre_hooks = OrderedDict()
    self._state_dict_hooks = OrderedDict()
    self._load_state_dict_pre_hooks = OrderedDict()
    self._modules = OrderedDict()
```

- nn.Module을 호출하면 이와 같은 것들이 초기화 된다.
- 대부분 OrderedDict 자료구조로 저장되어 있다.

forward



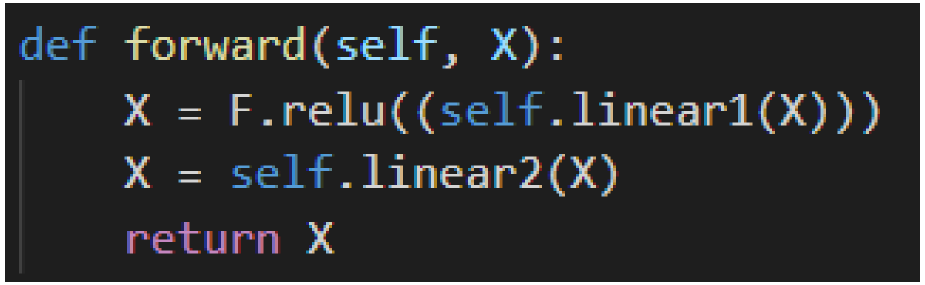
```
forward: Callable[..., Any] = _forward_unimplemented
```



```
def _forward_unimplemented(self, *input: Any) -> None:
    r"""Defines the computation performed at every call.

    Should be overridden by all subclasses.

    .. note::
        Although the recipe for forward pass needs to be defined within
        this function, one should call the :class:`Module` instance afterwards
        instead of this since the former takes care of running the
        registered hooks while the latter silently ignores them.
    """
    raise NotImplementedError
```

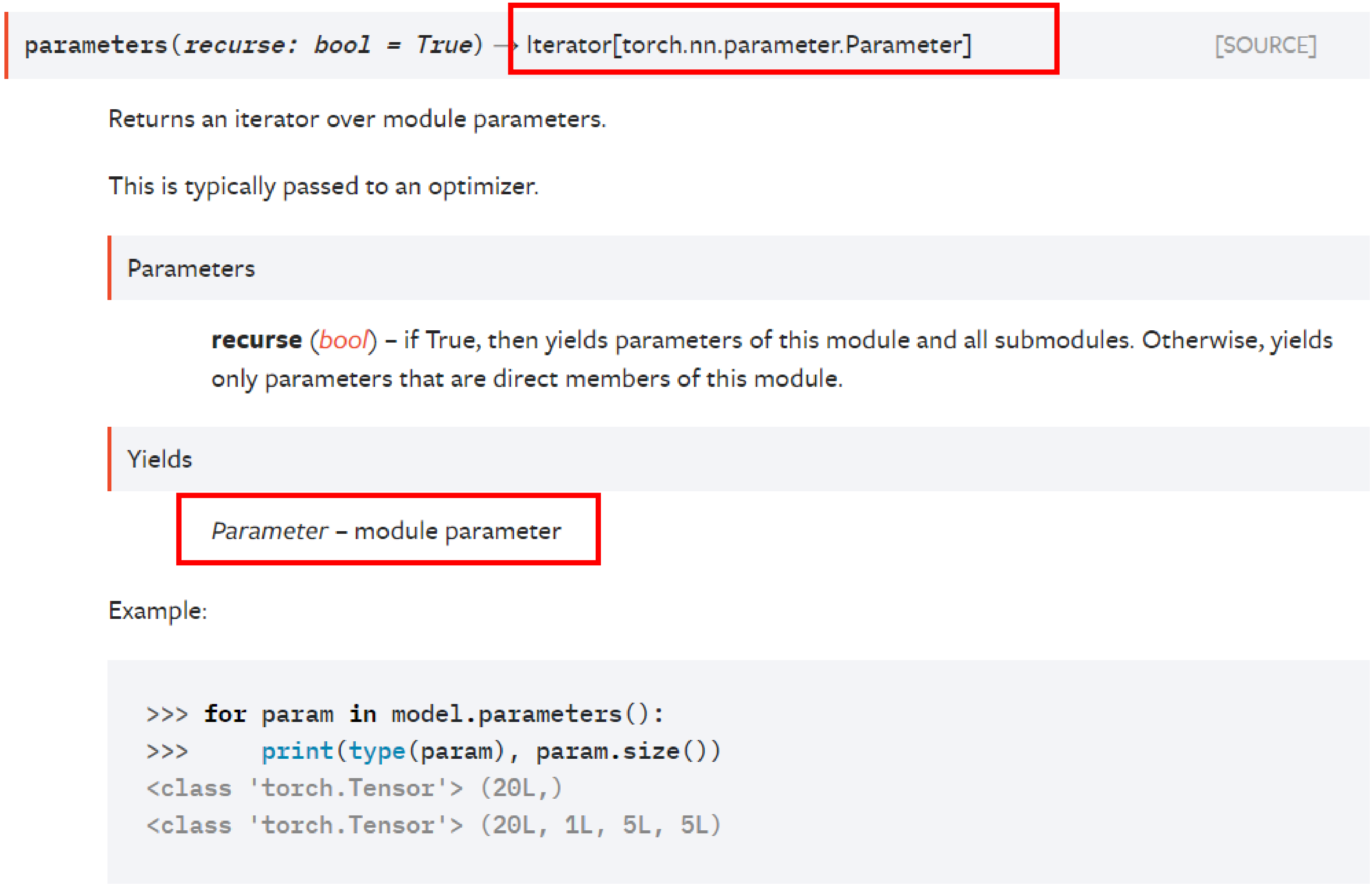
- nn.Module에 존재하는 forward에서는 입력으로 들어온 input이 Implemented 가능한지를 확인하고 불가능하면 예외를 발생시킨다.
- 실제로 어떤 layer를 거쳐서 forward를 진행할 것인지는 subclass에서 overridden을 해야한다.

강의 코드 예시



```
def forward(self, X):
    X = F.relu((self.linear1(X)))
    X = self.linear2(X)
    return X
```

parameters()



parameters(*recurse: bool = True*) → Iterator[torch.nn.parameter.Parameter]

[SOURCE]

Returns an iterator over module parameters.

This is typically passed to an optimizer.

Parameters

recurse (*bool*) – if True, then yields parameters of this module and all submodules. Otherwise, yields only parameters that are direct members of this module.

Yields

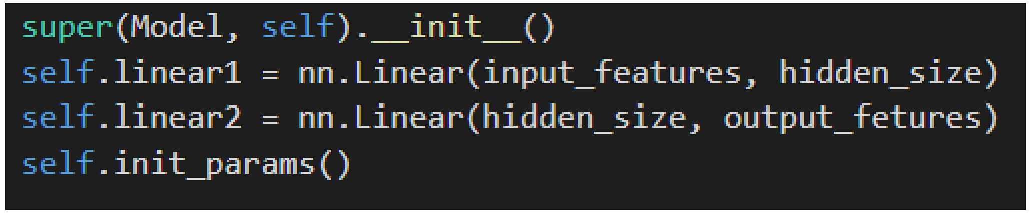
Parameter – module parameter

Example:

```
>>> for param in model.parameters():
>>>     print(type(param), param.size())
<class 'torch.Tensor'> (20L,)
<class 'torch.Tensor'> (20L, 1L, 5L, 5L)
```

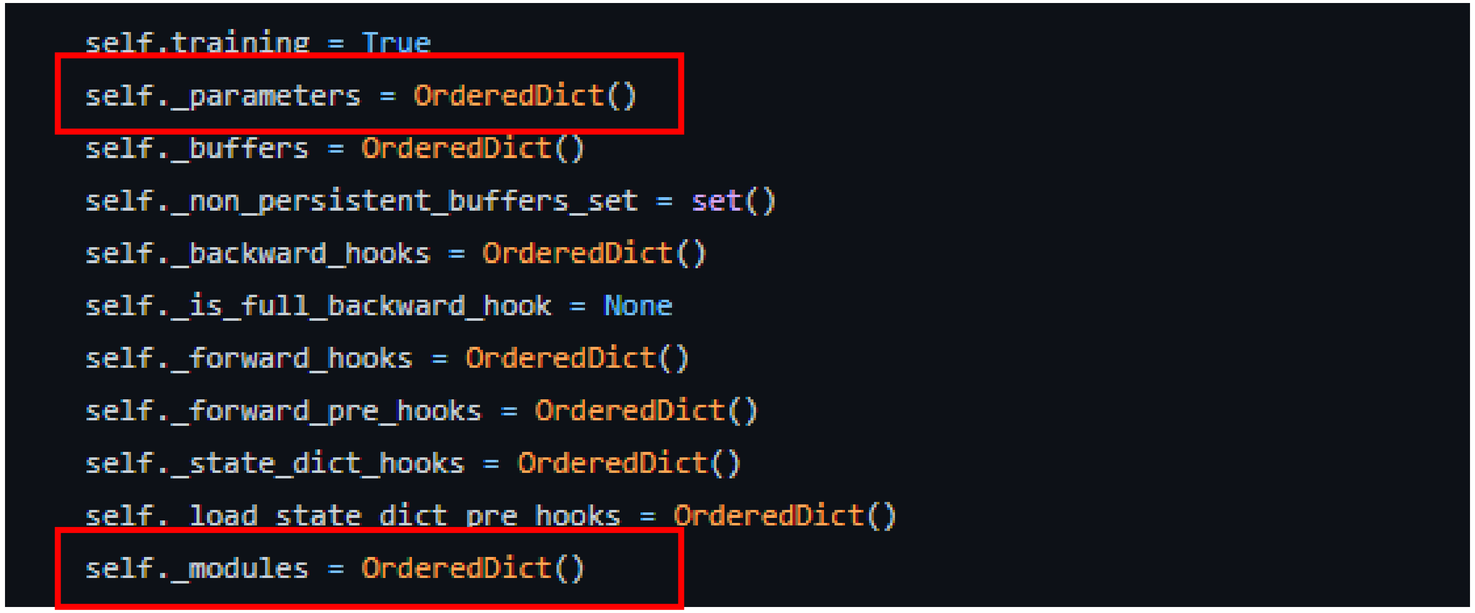
- Parameters()를 호출하면 모듈의 파라미터를 iterator로 반환
- Yields는 generator를 반환할 때 return 대신 사용되는 키워드

parameters()



```
super(Model, self).__init__()
self.linear1 = nn.Linear(input_features, hidden_size)
self.linear2 = nn.Linear(hidden_size, output_fetures)
self.init_params()
```

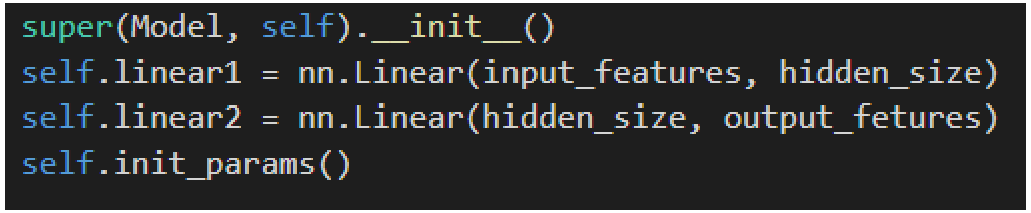
- 모듈의 파라미터는 무엇인가. 우리는 파라미터를 직접 추가하지 않는데?



```
self.training = True
self._parameters = OrderedDict()
self._buffers = OrderedDict()
self._non_persistent_buffers_set = set()
self._backward_hooks = OrderedDict()
self._is_full_backward_hook = None
self._forward_hooks = OrderedDict()
self._forward_pre_hooks = OrderedDict()
self._state_dict_hooks = OrderedDict()
self._load_state_dict_pre_hooks = OrderedDict()
self._modules = OrderedDict()
```

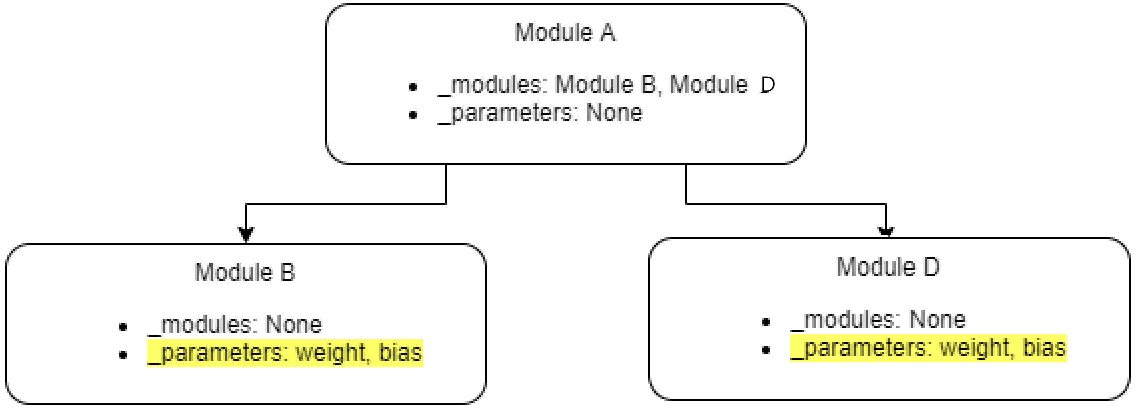
- Module을 __init__할때 자동으로 파라미터와 모듈이 설정된다.

parameters()



```
super(Model, self).__init__()
self.linear1 = nn.Linear(input_features, hidden_size)
self.linear2 = nn.Linear(hidden_size, output_fetures)
self.init_params()
```

- 모듈의 파라미터는 무엇인가. 우리는 파라미터를 직접 추가하지 않는데?



Module A

- _modules: Module B, Module D
- _parameters: None

Module B

- _modules: None
- _parameters: weight, bias

Module D

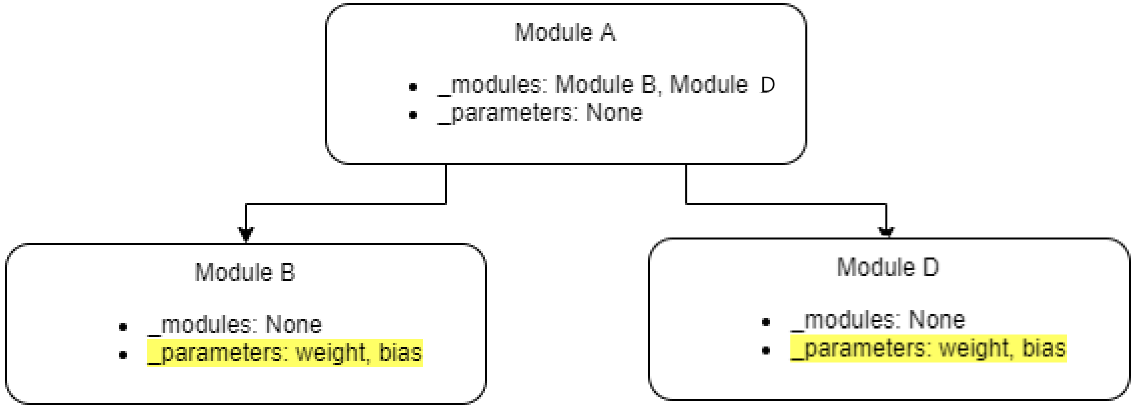
- _modules: None
- _parameters: weight, bias

Module A : MLP network
Module B : linear1
Module D : linear2

parameters()

- MLP에서 parameters()를 호출하면 [A.B weight, A.B bias, A.D weight, A.D bias]가 반환됨

- 그런데 parameters(recursive = False)로 지정하면 Module A내의 parameter만 반환하여 이 경우 아무것도 반환되지 않음



Module A

- _modules: Module B, Module D
- _parameters: None

Module B

- _modules: None
- _parameters: weight, bias

Module D

- _modules: None
- _parameters: weight, bias

Reference

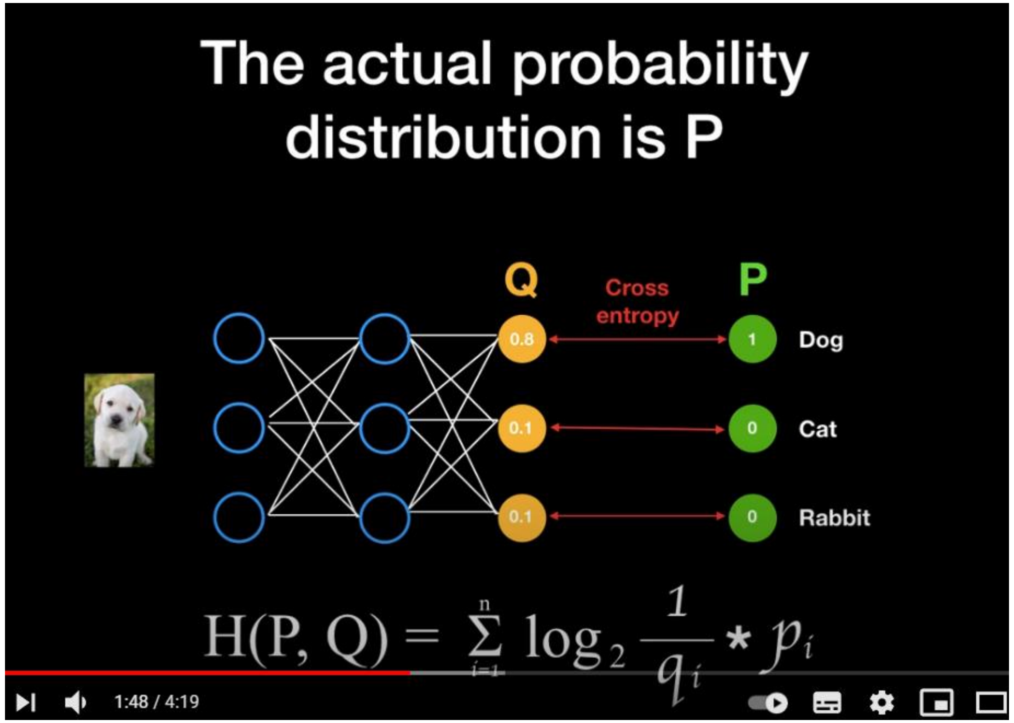
- Pytorch docs nn.Module
- https://pytorch.org/docs/stable/generated/torch.nn.Module.html

- Module class의 source code
- https://github.com/pytorch/pytorch/blob/master/torch/nn/modules/module.py

- Parameters()
- https://easy-going-programming.tistory.com/11?category=919874

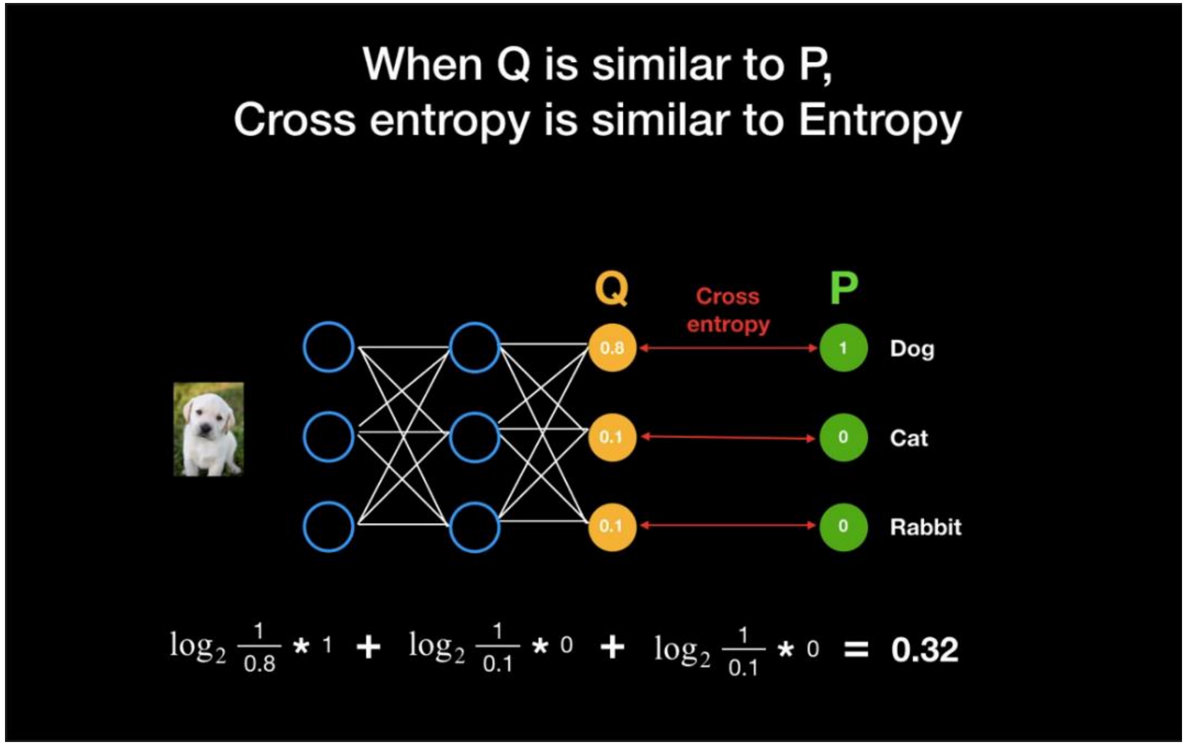
cross entropy

cross entropy 의 의미



The actual probability distribution is P

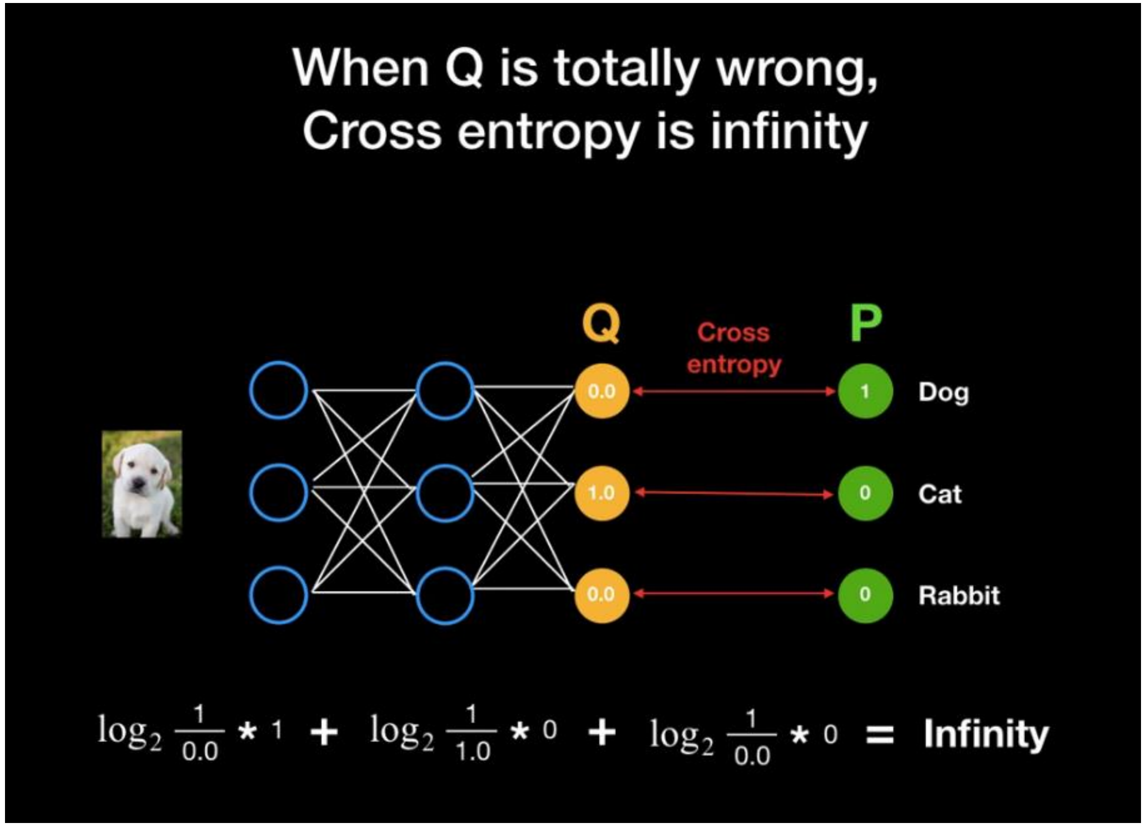
$$H(P, Q) = \sum_{i=1}^{n} \log_2 \frac{1}{q_i} * p_i$$



When Q is similar to P,
Cross entropy is similar to Entropy

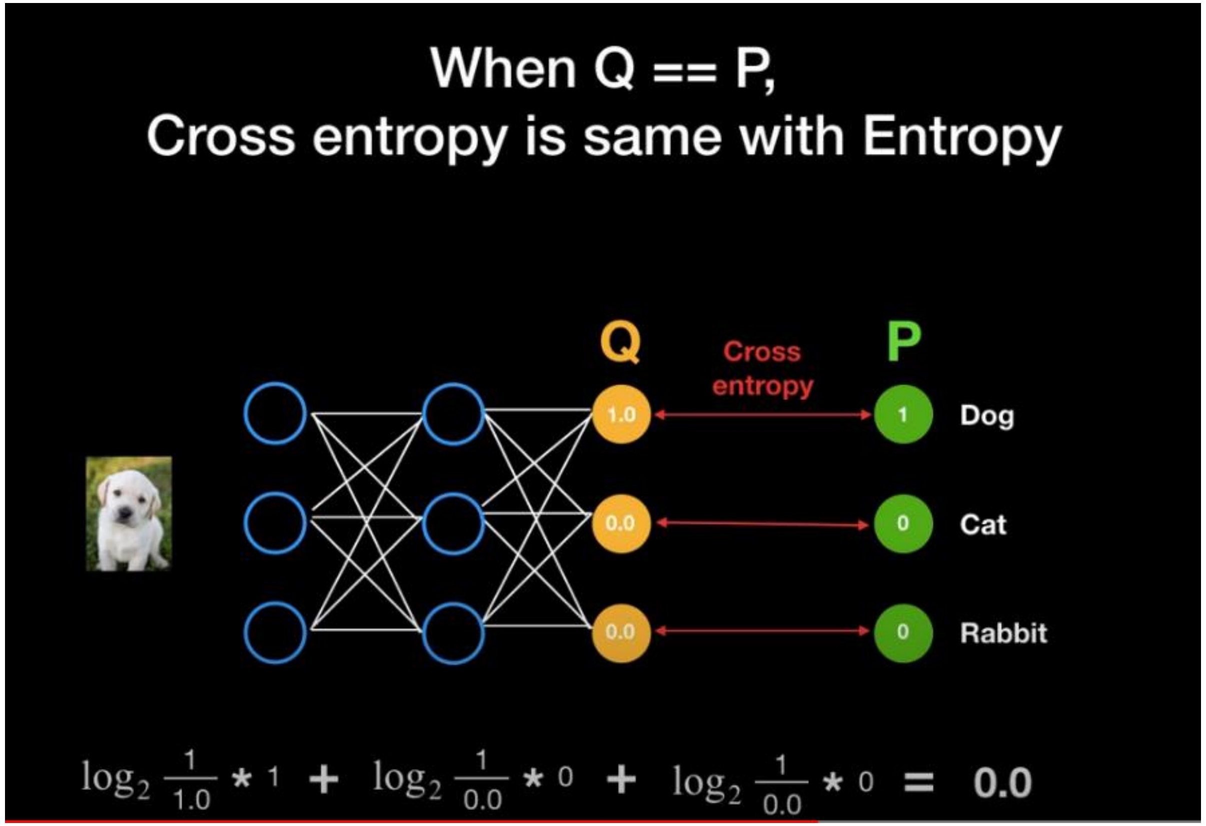
$$\log_2 \frac{1}{0.8} * 1 + \log_2 \frac{1}{0.1} * 0 + \log_2 \frac{1}{0.1} * 0 = 0.32$$

https://www.youtube.com/watch?v=Jt5BS71uVfI



When Q is totally wrong, Cross entropy is infinity

$$\log_2 \frac{1}{0.0} * 1 + \log_2 \frac{1}{1.0} * 0 + \log_2 \frac{1}{0.0} * 0 = \textbf{Infinity}$$

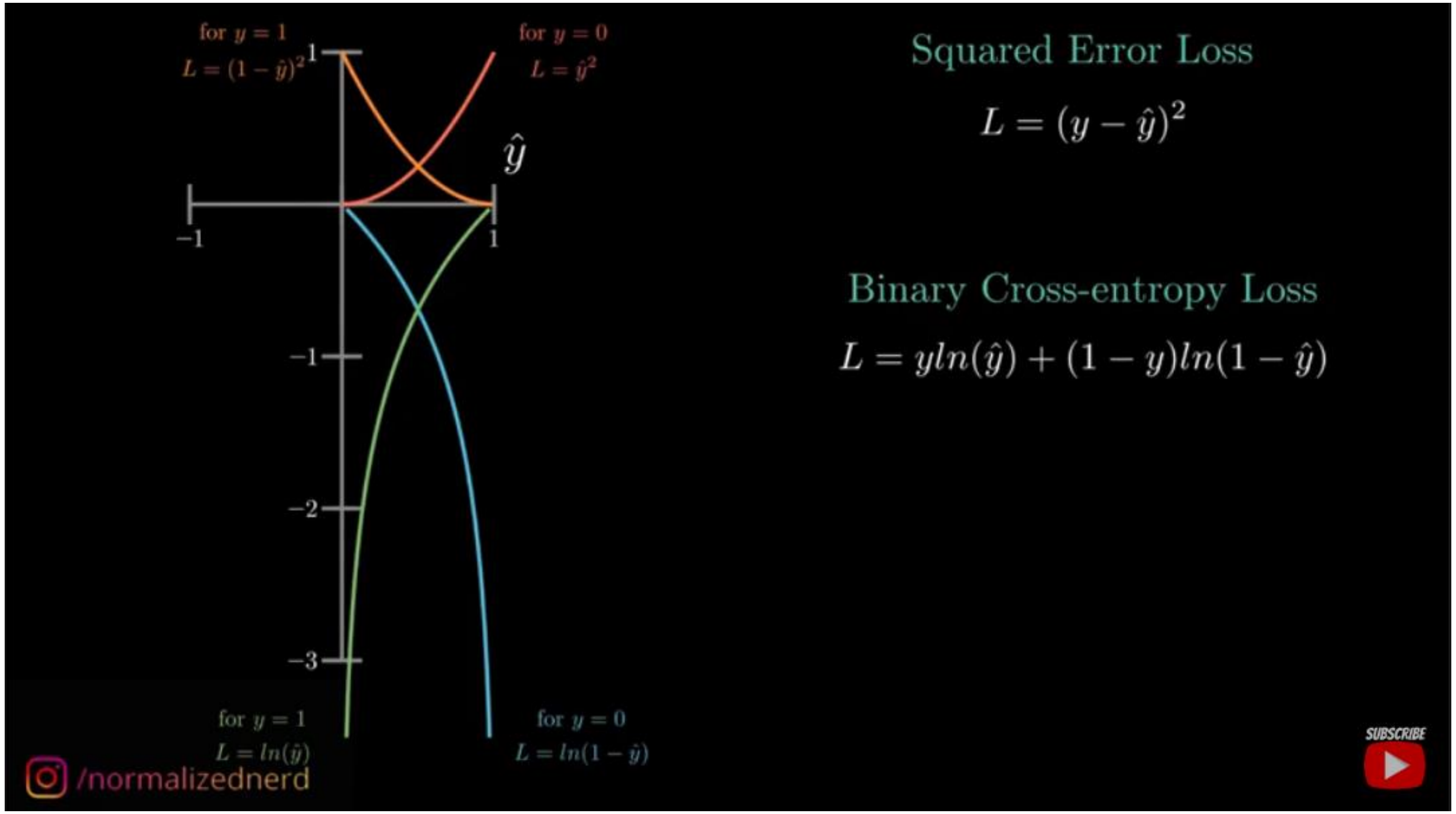


When Q == P, Cross entropy is same with Entropy

$$\log_2 \frac{1}{1.0} * 1 + \log_2 \frac{1}{0.0} * 0 + \log_2 \frac{1}{0.0} * 0 = \textbf{0.0}$$

cross entropy를 분류 문제에서 쓰는 이유

SE (square error loss) 와의 비교



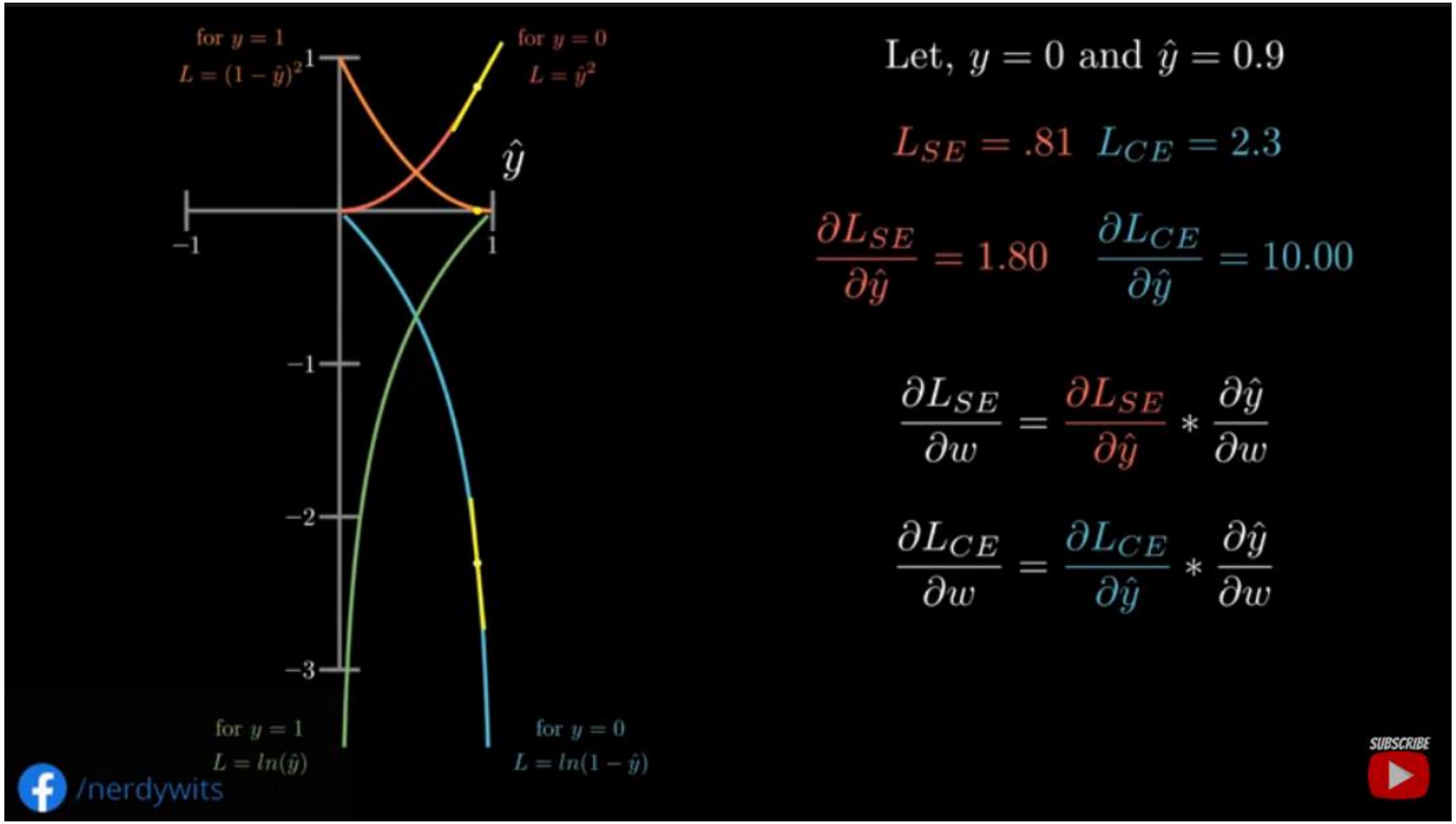
Squared Error Loss

$$L = (y - \hat{y})^2$$

Binary Cross-entropy Loss

$$L = yln(\hat{y}) + (1 - y)ln(1 - \hat{y})$$

https://www.youtube.com/watch?v=ErfnhcEV1O8



Let, $y = 0$ and $\hat{y} = 0.9$

$L_{SE} = .81 \quad L_{CE} = 2.3$

$$\frac{\partial L_{SE}}{\partial \hat{y}} = 1.80 \quad \frac{\partial L_{CE}}{\partial \hat{y}} = 10.00$$

$$\frac{\partial L_{SE}}{\partial w} = \frac{\partial L_{SE}}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial w}$$

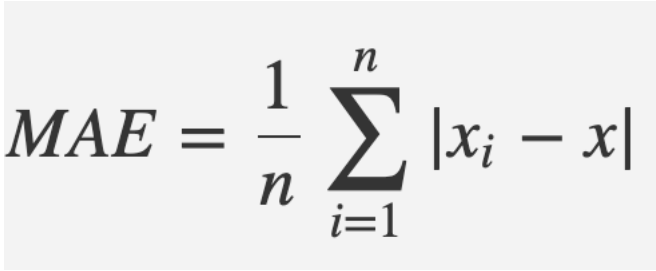
$$\frac{\partial L_{CE}}{\partial w} = \frac{\partial L_{CE}}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial w}$$

기울기의 차이 : 회귀문제와 달리 분류문제에서는 1이냐 0이냐가 중요하기 때문

Loss function

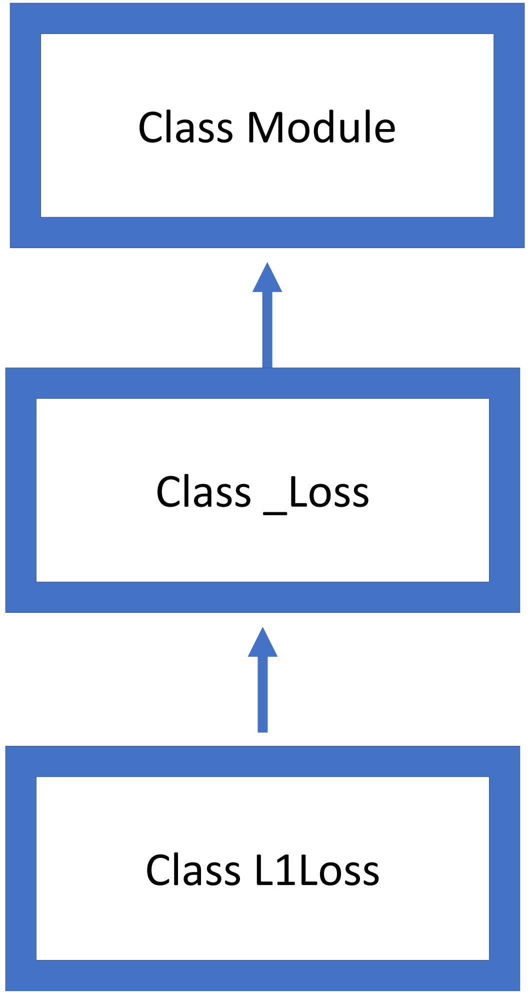
nn.L1Loss

- Creates a criterion that measures the mean absolute error (MAE) between each element in the input *x* and target *y* .



$$MAE = \frac{1}{n}\sum_{i=1}^{n} |x_i - x|$$

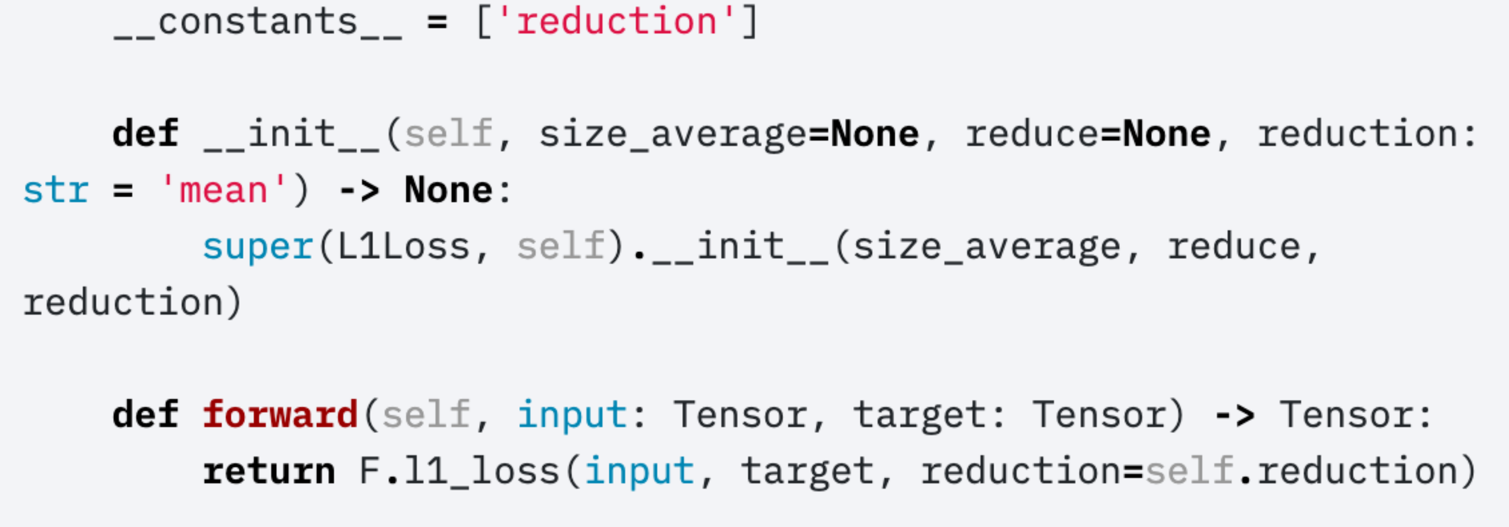
전체적인 구조



L1Loss 소스코드

$$\ell(x, y) = \begin{cases} \operatorname{mean}(L), & \text{if reduction} = \text{'mean';} \\ \operatorname{sum}(L), & \text{if reduction} = \text{'sum'.} \end{cases}$$

- **class L1Loss**(_Loss):

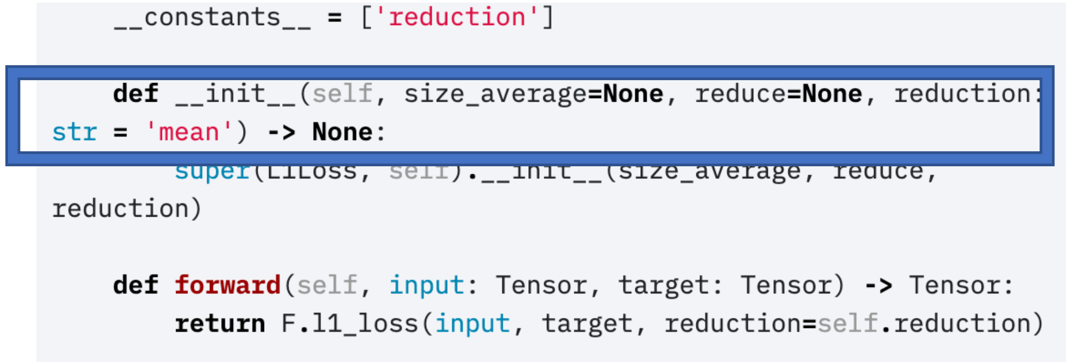


```
    __constants__ = ['reduction']

    def __init__(self, size_average=None, reduce=None, reduction:
str = 'mean') -> None:
        super(L1Loss, self).__init__(size_average, reduce,
reduction)

    def forward(self, input: Tensor, target: Tensor) -> Tensor:
        return F.l1_loss(input, target, reduction=self.reduction)
```

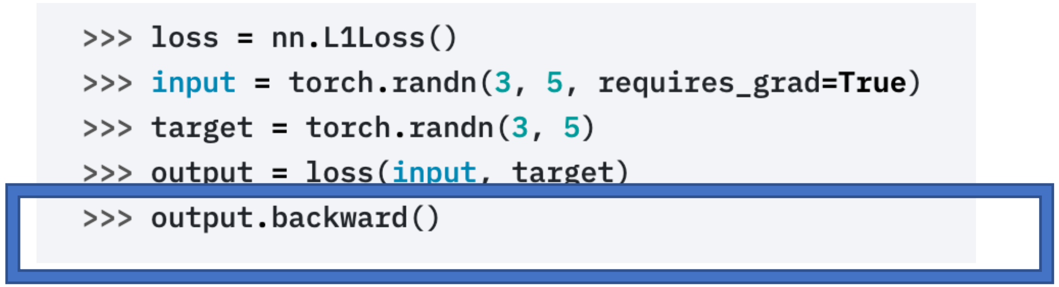
Ah yeah, the case you're seeing here is a hack that we've been using for a while, basically some modules can have optional submodules (i.e. either the submodule can be present or None). When we script something we just so happen to add submodules first, then constants, , skipping any names that are already present on the module, so it either gets added as a normal submodule (ignoring the entry in __constants__) or as a None constant. If the compiler sees a None constant in an if-statement it will skip compilation of the code inside the if, allowing us to support uses like downsample in resnet 15.

So **if you see a nn.Module in __constants__, all it really means is Optional[nn.Module], we just have this kind-of-nonsense way to specify that.**

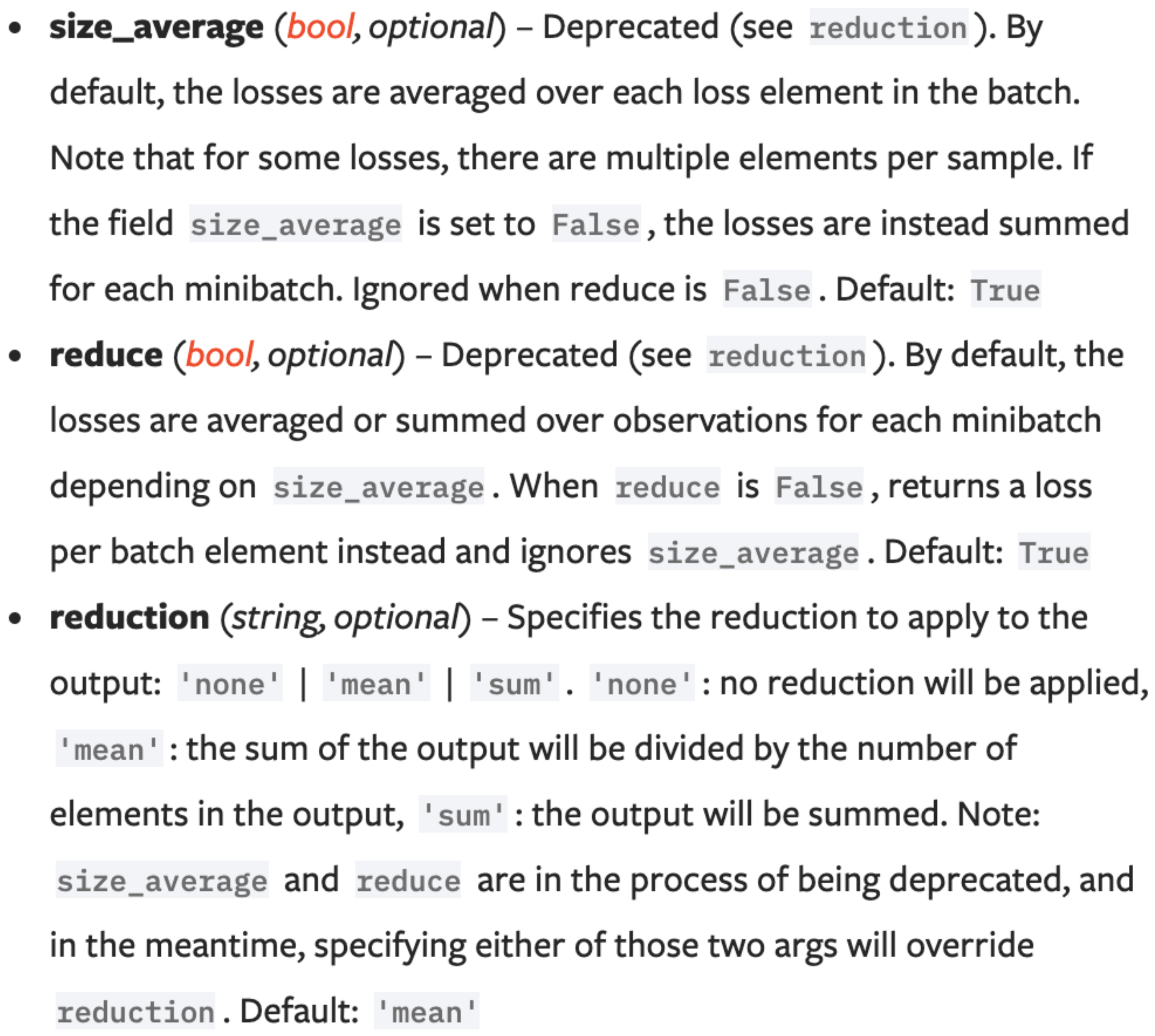
L1Loss 파라미터



```
    __constants__ = ['reduction']

    def __init__(self, size_average=None, reduce=None, reduction:
str = 'mean') -> None:
        super(L1Loss, self).__init__(size_average, reduce,
reduction)

    def forward(self, input: Tensor, target: Tensor) -> Tensor:
        return F.l1_loss(input, target, reduction=self.reduction)
```

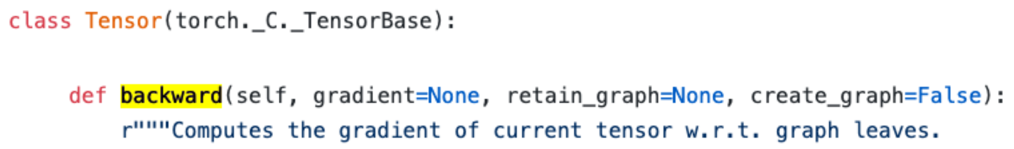


```
>>> loss = nn.L1Loss()
>>> input = torch.randn(3, 5, requires_grad=True)
>>> target = torch.randn(3, 5)
>>> output = loss(input, target)
>>> output.backward()
```

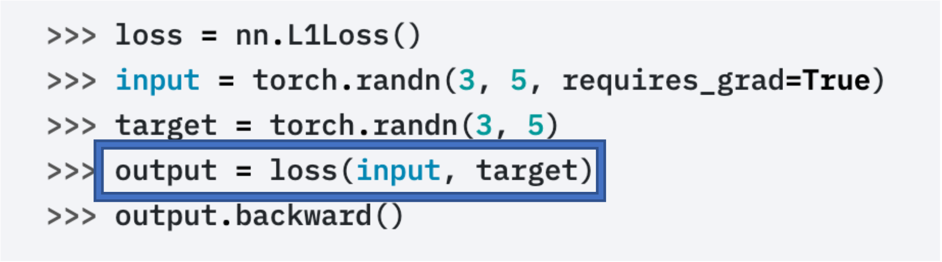


- **size_average** (*bool*, *optional*) – Deprecated (see `reduction`). By default, the losses are averaged over each loss element in the batch. Note that for some losses, there are multiple elements per sample. If the field `size_average` is set to `False`, the losses are instead summed for each minibatch. Ignored when reduce is `False`. Default: `True`
- **reduce** (*bool*, *optional*) – Deprecated (see `reduction`). By default, the losses are averaged or summed over observations for each minibatch depending on `size_average`. When `reduce` is `False`, returns a loss per batch element instead and ignores `size_average`. Default: `True`
- **reduction** (*string*, *optional*) – Specifies the reduction to apply to the output: `'none'` | `'mean'` | `'sum'`. `'none'`: no reduction will be applied, `'mean'`: the sum of the output will be divided by the number of elements in the output, `'sum'`: the output will be summed. Note: `size_average` and `reduce` are in the process of being deprecated, and in the meantime, specifying either of those two args will override `reduction`. Default: `'mean'`

Backward?



```
class Tensor(torch._C._TensorBase):

    def backward(self, gradient=None, retain_graph=None, create_graph=False):
        r"""Computes the gradient of current tensor w.r.t. graph leaves.
```



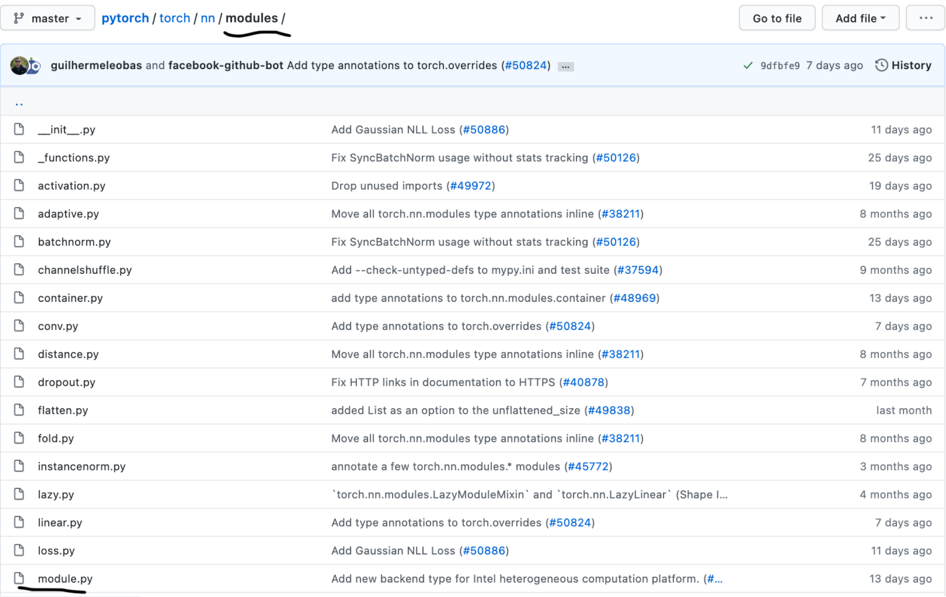
```
>>> loss = nn.L1Loss()
>>> input = torch.randn(3, 5, requires_grad=True)
>>> target = torch.randn(3, 5)
>>> output = loss(input, target)
>>> output.backward()
```

- [참조](https://datascience.stackexchange.com/questions/77390/where-is-the-backward-function-defined-in-pytorch)

_Loss 소스코드 (부모객체)

```python
from .distance import PairwiseDistance
from .module import Module
from .. import functional as F
from .. import _reduction as _Reduction
```

```python
class _Loss(Module):
    reduction: str

    def __init__(self, size_average=None, reduce=None, reduction:
str = 'mean') -> None:
        super(_Loss, self).__init__()
        if size_average is not None or reduce is not None:
            self.reduction =
 _Reduction.legacy_get_string(size_average, reduce)
        else:
            self.reduction = reduction
```



pytorch / torch / nn / modules /

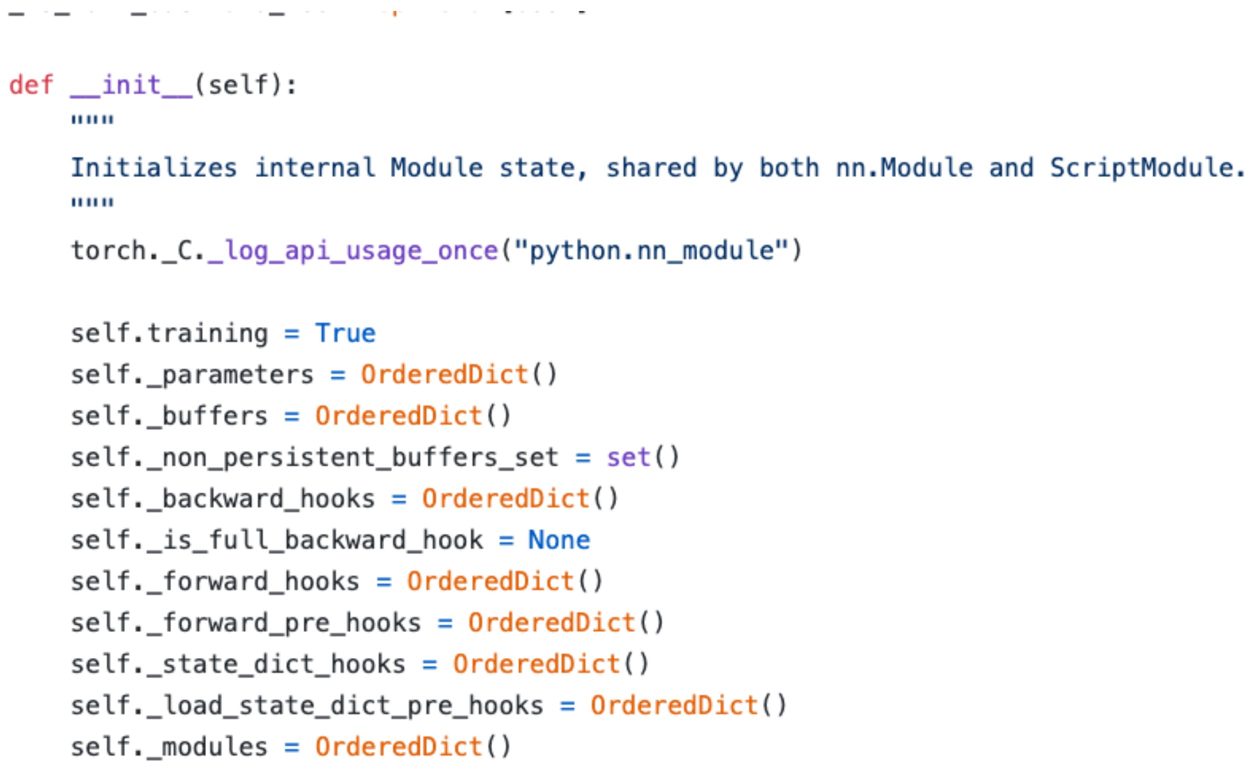
guilhermeleobas and facebook-github-bot Add type annotations to torch.overrides (#50824) … ✓ 9dfbfe9 7 days ago

..

__init__.py	Add Gaussian NLL Loss (#50886)	11 days ago
_functions.py	Fix SyncBatchNorm usage without stats tracking (#50126)	25 days ago
activation.py	Drop unused imports (#49972)	19 days ago
adaptive.py	Move all torch.nn.modules type annotations inline (#38211)	8 months ago
batchnorm.py	Fix SyncBatchNorm usage without stats tracking (#50126)	25 days ago
channelshuffle.py	Add --check-untyped-defs to mypy.ini and test suite (#37594)	9 months ago
container.py	add type annotations to torch.nn.modules.container (#48969)	13 days ago
conv.py	Add type annotations to torch.overrides (#50824)	7 days ago
distance.py	Move all torch.nn.modules type annotations inline (#38211)	8 months ago
dropout.py	Fix HTTP links in documentation to HTTPS (#40878)	7 months ago
flatten.py	added List as an option to the unflattened_size (#49838)	last month
fold.py	Move all torch.nn.modules type annotations inline (#38211)	8 months ago
instancenorm.py	annotate a few torch.nn.modules.* modules (#45772)	3 months ago
lazy.py	`torch.nn.modules.LazyModuleMixin` and `torch.nn.LazyLinear` (Shape I…	4 months ago
linear.py	Add type annotations to torch.overrides (#50824)	7 days ago
loss.py	Add Gaussian NLL Loss (#50886)	11 days ago
module.py	Add new backend type for Intel heterogeneous computation platform. (#…	13 days ago

Module 소스코드

- Base class for all neural network modules.



```
def __init__(self):
    """
    Initializes internal Module state, shared by both nn.Module and ScriptModule.
    """
    torch._C._log_api_usage_once("python.nn_module")

    self.training = True
    self._parameters = OrderedDict()
    self._buffers = OrderedDict()
    self._non_persistent_buffers_set = set()
    self._backward_hooks = OrderedDict()
    self._is_full_backward_hook = None
    self._forward_hooks = OrderedDict()
    self._forward_pre_hooks = OrderedDict()
    self._state_dict_hooks = OrderedDict()
    self._load_state_dict_pre_hooks = OrderedDict()
    self._modules = OrderedDict()
```

KL Divergence

Kullback-Leibler Divergence

- 두 개의 확률분포 P(x), Q(x) 사이의 거리 계산할 때 사용

이산확률변수 연속확률변수

- 정의: $\mathbb{KL}(P\|Q) = \sum_{\mathbf{x}\in\mathcal{X}} P(\mathbf{x}) \log\left(\frac{P(\mathbf{x})}{Q(\mathbf{x})}\right)$ $\mathbb{KL}(P\|Q) = \int_{\mathcal{X}} P(\mathbf{x}) \log\left(\frac{P(\mathbf{x})}{Q(\mathbf{x})}\right) \mathrm{d}\mathbf{x}$

- 분해: $\mathbb{KL}(P\|Q) = -\mathbb{E}_{\mathbf{x}\sim P(\mathbf{x})}[\log Q(\mathbf{x})] + \mathbb{E}_{\mathbf{x}\sim P(\mathbf{x})}[\log P(\mathbf{x})]$

크로스 엔트로피 엔트로피

- 분류 문제에서 정답 레이블: P, 모델 예측: Q
  →최대가능도 추정법: 쿨백-라이블러 발산 최소화

KLDivLoss

- torch.nn.KLDivLoss(*size_average=None, reduce=None, reduction: str = 'mean', log_target: bool = False*)
- class KLDivLoss(_Loss):

```
    __constants__ = ['reduction’]

    def __init__(self, size_average=None, reduce=None, reduction: str = 'mean', log_target: bool = False) -> None:
        super(KLDivLoss, self).__init__(size_average, reduce, reduction)
        self.log_target = log_target

    def forward(self, input: Tensor, target: Tensor) -> Tensor:
        return F.kl_div(input, target, reduction=self.reduction, log_target=self.log_target)
```

Parameters

- ~~**size_average** (*bool*, *optional*) – Deprecated (see reduction)~~
- ~~**reduce** (*bool*, *optional*) – Deprecated (see reduction)~~
- **reduction** (*string, optional*)
  - none: reduction 안 함

    $$l(x, y) = L = \{l_1, \dots, l_N\}, \quad l_n = y_n \cdot (\log y_n - x_n)$$

  - batchmean: output 합 / 배치 사이즈 (실제 KL Divergence)
  - sum: output 합
  - **mean**: output 합 / output 원소 개수(**default**)

    $$\ell(x, y) = \begin{cases} \mathrm{mean}(L), & \text{if reduction} = \text{'mean';} \\ \mathrm{sum}(L), & \text{if reduction} = \text{'sum'.} \end{cases}$$

- **log_target** (*bool*, *optional*) → Default: **False**

Shape

- Input: (N, *)
- Target: (N, *)
- Output
  - Default: scalar
  - reduction=‘none’일 때: (N, *) (input과 같음)

Reference

- https://pytorch.org/docs/stable/generated/torch.nn.KLDivLoss.html
- https://github.com/pytorch/pytorch/blob/master/torch/nn/modules/loss.py