O PyTorch

Q Search Docs

Notes [+] Language Bindings [+]

Python API [-] torch torch.nn

torch.nn.functional torch.Tensor

Tensor Attributes Tensor Views torch.autograd

torch.cuda torch.cuda.amp

torch.backends torch.distributed torch.distributed.algorithms.join torch.distributed.elastic torch.distributed.fsdp torch.distributed.optim

torch.distributions torch.fft torch.futures

torch.hub torch.jit torch.linalg torch.monitor torch.special torch.overrides torch.package torch.profiler

torch.fx

torch.nn.init torch.onnx

torch.optim

Docs > torch.nn > CrossEntropyLoss

Get Started

Mobile

CROSSENTROPYLOSS

Ecosystem

CLASS torch.nn.CrossEntropyLoss(weight=None, size_average=None, ignore_index=- 100, reduce=None, reduction='mean', label_smoothing=0.0) [SOURCE]

Blog

This criterion computes the cross entropy loss between input and target.

It is useful when training a classification problem with C classes. If provided, the optional argument weight should be a 1D Tensor assigning weight to each of the classes. This is particularly useful when you have an unbalanced training set.

Tutorials

Docs 🗸

GitHub

>_

Resources ~

The input is expected to contain raw, unnormalized scores for each class. input has to be a Tensor of size (C) for unbatched input, (minibatch,C) or $(minibatch,C,d_1,d_2,...,d_K)$ with $K\geq 1$ for the K-dimensional case. The last being useful for higher dimension inputs, such as computing cross entropy loss per-pixel for 2D images.

The target that this criterion expects should contain either:

ullet Class indices in the range [0,C) where C is the number of classes; if ignore_index is specified, this loss also accepts this class index (this index may not necessarily be in the class range). The unreduced (i.e. with reduction set to 'none') loss for this case can be described as:

$$\ell(x,y) = L = \{l_1,\ldots,l_N\}^ op, \quad l_n = -w_{y_n}\lograc{\exp(x_{n,y_n})}{\sum_{c=1}^C\exp(x_{n,c})}\cdot 1\{y_n
eq ext{ignore_index}\}$$

where x is the input, y is the target, w is the weight, C is the number of classes, and N spans the minibatch dimension as well as $d_1,...,d_k$ for the K-dimensional case. If reduction is not 'none' (default 'mean'), then

$$\ell(x,y) = egin{cases} \sum_{n=1}^N rac{1}{\sum_{n=1}^N w_{y_n} \cdot 1\{y_n
eq ext{ignore_index}\}} l_n, & ext{if reduction} = ext{`mean'}; \ \sum_{n=1}^N l_n, & ext{if reduction} = ext{`sum'}. \end{cases}$$

Note that this case is equivalent to the combination of LogSoftmax and NLLLoss.

• Probabilities for each class; useful when labels beyond a single class per minibatch item are required, such as for blended labels, label smoothing, etc. The unreduced (i.e. with reduction set to 'none') loss for this case can be described as:

$$\ell(x,y) = L = \{l_1,\dots,l_N\}^ op, \quad l_n = -\sum_{c=1}^C w_c \log rac{\exp(x_{n,c})}{\sum_{i=1}^C \exp(x_{n,i})} y_{n,c}$$

where x is the input, y is the target, w is the weight, C is the number of classes, and N spans the minibatch dimension as well as $d_1,...,d_k$ for the K-dimensional case. If reduction is not 'none' (default 'mean'), then

$$\ell(x,y) = egin{cases} rac{\sum_{n=1}^N l_n}{N}, & ext{if reduction} = ext{`mean'}; \ \sum_{n=1}^N l_n, & ext{if reduction} = ext{`sum'}. \end{cases}$$

• NOTE

The performance of this criterion is generally better when target contains class indices, as this allows for optimized computation. Consider providing target as class probabilities only when a single class label per minibatch item is too restrictive.

Parameters

- weight (Tensor, optional) a manual rescaling weight given to each class. If given, has to be a Tensor of size
- **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
- **ignore_index** (*int*, *optional*) Specifies a target value that is ignored and does not contribute to the input gradient. When size_average is True, the loss is averaged over non-ignored targets. Note that ignore_index is only applicable when the target contains class indices.
- 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 weighted mean of the output is taken, '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'
- label_smoothing (float, optional) A float in [0.0, 1.0]. Specifies the amount of smoothing when computing the loss, where 0.0 means no smoothing. The targets become a mixture of the original ground truth and a uniform distribution as described in Rethinking the Inception Architecture for Computer Vision. Default: 0.0.

Shape:

- ullet Input: Shape (C), (N,C) or $(N,C,d_1,d_2,...,d_K)$ with $K\geq 1$ in the case of K-dimensional loss.
- ullet Target: If containing class indices, shape () , (N) or $(N,d_1,d_2,...,d_K)$ with $K\geq 1$ in the case of Kdimensional loss where each value should be between [0,C). If containing class probabilities, same shape as the input and each value should be between [0,1].
- Output: If reduction is 'none', same shape as the target. Otherwise, scalar.

where:

C = number of classes

N =batch size

Examples:

>>> # Example of target with class indices >>> loss = nn.CrossEntropyLoss() >>> input = torch.randn(3, 5, requires_grad=True) >>> target = torch.empty(3, dtype=torch.long).random_(5) >>> output = loss(input, target) >>> output.backward() >>> # Example of target with class probabilities >>> input = torch.randn(3, 5, requires_grad=True) >>> target = torch.randn(3, 5).softmax(dim=1) >>> output = loss(input, target) >>> output.backward()

Previous Next >

© Copyright 2019, Torch Contributors.

Built with Sphinx using a theme provided by Read the Docs.

Docs **Tutorials** Resources

Access comprehensive developer documentation for Get in-depth tutorials for beginners and advanced

PyTorch developers Find development resources and get your questions answered

View Tutorials > View Docs > View Resources >

Contributing

PyTorch Resources Get Started **Tutorials** Docs

Features Ecosystem Discuss Blog Github Issues

Brand Guidelines

6 9 0

Stay Connected

Email Address

 \rightarrow