

CROSSENTROPYLOSS

CLASS

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

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.

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 $(minibatch, C, d_1, d_2, \dots, d_K)$ 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:

- 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:

$$\ell(x, y) = L = \{l_1, \dots, l_N\}^\top, \quad l_n = -w_{y_n} \log \frac{\exp(x_{n,y_n})}{\sum_{c=1}^C \exp(x_{n,c})} \cdot 1\{y_n \neq \text{ignore_index}\}$$

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, \dots, d_k for the K -dimensional case. If `reduction` is not `'none'` (default `'mean'`), then

$$\ell(x, y) = \begin{cases} \sum_{n=1}^N \frac{1}{\sum_{n=1}^N w_{y_n} \cdot 1\{y_n \neq \text{ignore_index}\}} l_n, & \text{if reduction = 'mean'}; \\ \sum_{n=1}^N l_n, & \text{if reduction = 'sum'}. \end{cases}$$

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:

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

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, \dots, d_k for the K -dimensional case. If `reduction` is not `'none'` (default `'mean'`), then

$$\ell(x, y) = \begin{cases} \frac{\sum_{n=1}^N l_n}{N}, & \text{if reduction = 'mean'}; \\ \sum_{n=1}^N l_n, & \text{if reduction = 'sum'}. \end{cases}$$

• 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
<ul style="list-style-type: none">weight (<i>Tensor, optional</i>) – a manual rescaling weight given to each class. If given, has to be a Tensor of size Csize_average (<i>bool, optional</i>) – Deprecated (see <code>reduction</code>). 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 <code>size_average</code> is set to <code>False</code>, the losses are instead summed for each minibatch. Ignored when <code>reduce</code> is <code>False</code>. Default: <code>True</code>ignore_index (<i>int, optional</i>) – Specifies a target value that is ignored and does not contribute to the input gradient. When <code>size_average</code> is <code>True</code>, the loss is averaged over non-ignored targets. Note that <code>ignore_index</code> is only applicable when the target contains class indices.reduce (<i>bool, optional</i>) – Deprecated (see <code>reduction</code>). By default, the losses are averaged or summed over observations for each minibatch depending on <code>size_average</code>. When <code>reduce</code> is <code>False</code>, returns a loss per batch element instead and ignores <code>size_average</code>. Default: <code>True</code>reduction (<i>string, optional</i>) – Specifies the reduction to apply to the output: <code>'none'</code> <code>'mean'</code> <code>'sum'</code>. <code>'none'</code>: no reduction will be applied, <code>'mean'</code>: the weighted mean of the output is taken, <code>'sum'</code>: the output will be summed. Note: <code>size_average</code> and <code>reduce</code> are in the process of being deprecated, and in the meantime, specifying either of those two args will override <code>reduction</code>. Default: <code>'mean'</code>label_smoothing (<i>float, optional</i>) – 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: <code>0.0</code>.
Shape:
<ul style="list-style-type: none">Input: Shape (C), (N, C) or $(N, C, d_1, d_2, \dots, d_K)$ with $K \geq 1$ in the case of K-dimensional loss.Target: If containing class indices, shape $()$, (N) or $(N, d_1, d_2, \dots, d_K)$ with $K \geq 1$ in the case of K-dimensional 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 ▶](#)

Docs

Access comprehensive developer documentation for
PyTorch
[View Docs](#)

Tutorials

Get in-depth tutorials for beginners and advanced
developers
[View Tutorials](#)

Resources

Find development resources and get your questions
answered
[View Resources](#)

PyTorch

Get Started

Features

Ecosystem

Blog

Contributing

Resources

Tutorials

Docs

Discuss

Github Issues

Brand Guidelines

Stay Connected