cross entropy

강의찬

```
In [26]: for epoch in range(10):
             model, train_loss = train(model, train_loader, optimizer)
             test_loss, test_accuracy = evaluate(model, test_loader)
             print('[{} Test Loss : {:.4f}, Accuracy : {:.2f}%]'.format(epoch, test_loss, test_accuracy))
```

```
output tensor([[0.4901, 0.5024, 0.6291]], device='cuda:0', grad_fn=<SigmoidBackward>)
target tensor([1], device='cuda:0')
output tensor([[0.4879, 0.5221, 0.5935]], device='cuda:0', grad_fn=<SigmoidBackward>)
target tensor([0], device='cuda:0')
output tensor([[0.4957, 0.5267, 0.5839]], device='cuda:0', grad_fn=<SigmoidBackward>)
target tensor([0], device='cuda:0')
output tensor([[0.4933, 0.5412, 0.5688]], device='cuda:0', grad_fn=<SigmoidBackward>)
target tensor([1], device='cuda:0')
output tensor([[0.4932, 0.5584, 0.5552]], device='cuda:0', grad_fn=<SigmoidBackward>)
target tensor([2], device='cuda:0')
output tensor([[0.5098, 0.5306, 0.5523]], device='cuda:0', grad_fn=<SigmoidBackward>)
target tensor([0], device='cuda:0')
output tensor([[0.5116, 0.5335, 0.5529]], device='cuda:0', grad_fn=<SigmoidBackward>)
target tensor([0], device='cuda:0')
output tensor([[0.5014, 0.5512, 0.5415]], device='cuda:0', grad_fn=<SigmoidBackward>)
target tensor([1], device='cuda:0')
output tensor([[0.5015, 0.5608, 0.5357]], device='cuda:0', grad_fn=<SigmoidBackward>)
target tensor([1], device='cuda:0')
output tensor([[0.5230, 0.5307, 0.5322]], device='cuda:0', grad_fn=<SigmoidBackward>)
target tensor([0], device='cuda:0')
```

Loss 함수에서 사용되는 output과 targe의 shape가 다르다?
⇒ 구글링

richard Feb '18

nn.CrossEntropyLoss 371 doesn't take a one-hot vector, it takes class values. You can create a new function that wraps `nn.CrossEntropyLoss`, in the following manner:

```
def cross_entropy_one_hot(input, target):
    _, labels = target.max(dim=0)
    return nn.CrossEntropyLoss()(input, labels)
```

Also I'm not sure I'm understanding what you want. nn.BCELossWithLogits 212 and nn.CrossEntropyLoss are different in the docs; I'm not sure in what situation you would expect the same loss from them.

outpu과 label을 one-hot vecto로 비교하지 않는다?
=> cross_entropy 함수 설명

https://discuss.pytorch.org/t/cross-entropy-with-one-hot-targets/13580/3

CrossEntropyLoss

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

This criterion combines `nn.LogSoftmax()` and `nn.NLLLoss()` in one single class.

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 either $(minibatch, C)$ or $(minibatch, C, d_1, d_2, ..., d_K)$ with $K \geq 1$ for the *K*-dimensional case (described later).

This criterion expects a class index in the range $[0, C-1]$ as the *target* for each value of a 1D tensor of size *minibatch*; if *ignore_index* is specified, this criterion also accepts this class index (this index may not necessarily be in the class range).

The loss can be described as:

$$\text{loss}(x, class) = -\log\left(\frac{\exp(x[class])}{\sum_j \exp(x[j])}\right) = -x[class] + \log\left(\sum_j \exp(x[j])\right)$$

or in the case of the `weight` argument being specified:

$$\text{loss}(x, class) = weight[class]\left(-x[class] + \log\left(\sum_j \exp(x[j])\right)\right)$$

The losses are averaged across observations for each minibatch.

Can also be used for higher dimension inputs, such as 2D images, by providing an input of size $(minibatch, C, d_1, d_2, ..., d_K)$ with $K \geq 1$, where K is the number of dimensions, and a target of appropriate shape (see below).

[0, class -1]의 값을 targe으로 넣는다.
=> 왜?

1. Information Theory

Entropy : 불확실성에 대한 척도, 놀람의 정도

Entropy = $-\sum_i P_i \log P_i$

Example

- 가위 바위 보

  – 두 사람이 동등한 이길 확률을 가질 때 $E = -[0.5\log(0.5) + 0.5\log(0.5)]$ = 0.69

  - 한 명의 이길 확률이 더 높을 때 $E = -[0.1\log(0.1) + 0.9\log(0.9)]$ = 0.32

  – 한 명이 무조건 이길 때 $E = -[1\log 1(1)]$ = 0

  -> 하나의 사건의 확률이 증가할 수록 Entropy는 작아진다.(=놀랍지 않다.)

2. Cross Entropy

두 개의 확률 분포 p와 q에 대해 하나의 사건 X가 갖는 정보량.

$$H_{p,q}(X) = -\sum_{x} p(x) \log q(x)$$

$$(\; p: \; probability \;\; of \;\; underlying \;\; true \;\; density,$$
$$q: \; probability \;\; of \;\; parametric \;\; model \;)$$

두 확률 분포 p, q 사이에 존재하는 정보량을 계산하는 방법.
정확히는 q에 대한 정보량을 p에 대해서 평균낸 것이다.

출처: https://newsight.tistory.com/119

3. Kullback-Leibler Divergence

$$D_{KL}(p||q) = \sum_x p(x) \log \frac{p(x)}{q(x)}$$

두 확률 분포 p, q 간의 거리를 측정하는 방법
p(x)와 q(x)의 값이 항상 같을 경우, 정보량에 해당하는 부분은 항상 값이 1이 된다.
이 값이 1이되면 D_KL값은 항상 0이 되고, 두 분포간의 거리가 0인 것을 의미한다.

출처: https://newsight.tistory.com/119

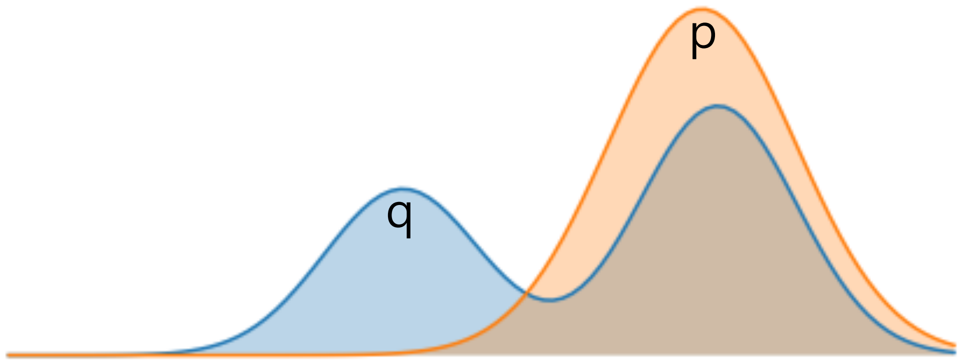
3. Kullback-Leibler Divergence

$$
\begin{aligned}
D_{KL}(p||q) &= \sum_x p(x)\log\frac{p(x)}{q(x)} \\
&= \sum_x p(x)\log p(x) - \sum_x p(x)\log q(x) \\
&= -H_p(X) + H_{p,q}(X)
\end{aligned}
$$

$$
H_{p,q}(X) = D_{KL}(p||q) + H_p(X)
$$

확률 분포 p와 q의 cross entropy는 KLD와 확률 분포 p의 entropy로 구할 수 있다.
p는 항상 상수(여기서는 0)이기 때문에 KLD를 최소화시키면 두 확률 분포가 같아진다.

3. Kullback-Leibler Divergence



고정되어 있는 확률 분포 p(주황색)과 예측 확률 분포 q(파란색)이 있을 때,
q를 p와 같은 모양으로 만들면, 거리가 KLD가 0이 된다.

4. Cross Entropy Minimization

그런데 KLD와 확률 분포 p의 Entropy를 계산하는 것이 수식적으로 힘들다고 한다.

그래서 softmax와 NLLLoss를 사용해 CrossEntropyLoss를 구현한…다.

This criterion combines `nn.LogSoftmax()` and `nn.NLLLoss()` in one single class. (Negative Log Likelihood)

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 either $(minibatch, C)$ or $(minibatch, C, d_1, d_2, ..., d_K)$ with $K \geq 1$ for the *K*-dimensional case (described later).

This criterion expects a class index in the range $[0, C-1]$ as the *target* for each value of a 1D tensor of size *minibatch*; if *ignore_index* is specified, this criterion also accepts this class index (this index may not necessarily be in the class range).

The loss can be described as:

$$\text{loss}(x, class) = -\log\left(\frac{\exp(x[class])}{\sum_j \exp(x[j])}\right) = -x[class] + \log\left(\sum_j \exp(x[j])\right)$$

정보이론 : https://icim.nims.re.kr/post/easyMath/550

https://newsight.tistory.com/119

Softmax classifier : https://cs231n.github.io/linear-classify/#softmax-classifier