Understanding Categorical Cross-Entropy Loss, Binary Cross-Entropy Loss, Softmax Loss, Logistic Loss and all those confusing names

Introduction
Concepts
- 2.1 Tasks
  - 2.1.1 Multi-Class Classification
  - 2.1.2 Multi-Label Classification
- 2.2 Output Activation Functions
  - 2.2.1 Sigmoid
  - 2.2.2 Softmax

Losses
- 3.1 Cross-Entropy loss
- 3.2 Categorical Cross-Entropy loss
- 3.3Binary Cross-Entropy Loss

本篇文章的原文地址为：点我看原文

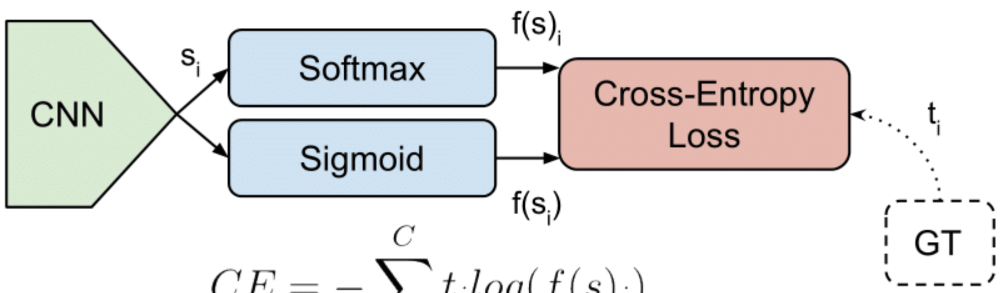
值得看的相关的文章：

- Pytorch里的CrossEntropyLoss详解，这篇文章主要说明了Pytorch里面CrossEntropyLoss函数的计算流程
- 多分类的交叉熵和二分类的交叉熵有什么联系？，这个知乎问题中的高赞回答与本文内容相似，但是里面的一些定义没说清楚导致我看得迷迷糊糊的，看完下面的内容后再反过头来看这篇知乎文章就很轻松了
- Python实战——VAE的理论详解及Pytorch实现，这里面非常详细地介绍了VAE的理论以及Pytorch实现的代码
- Understanding binary cross-entropy / log loss: a visual explanation，可视化损失函数的计算过程，比较适合新手

Introduction

People like to use cool names which are often confusing. When I started playing with CNN beyond single label classification, I got confused with the different names and formulations people write in their papers, and even with the loss layer names of the deep learning frameworks such as Caffe, Pytorch or TensorFlow.

In this post I group up the different names and variations people use for **Cross-Entropy Loss**. I explain their main points, use cases and the implementations in different deep learning frameworks.



$$CE = -\sum_{i}^{C} t_i log(f(s)_i)$$

$$CE = -\sum_{i=1}^{C'=2} t_i log(f(s_i)) = -t_1 log(f(s_1)) - (1 - t_1) log(1 - f(s_1))$$

Fig.1 两种激活函数下Cross Entropy的计算公式

First, let’s introduce some concepts:

Concepts

2.1 Tasks

2.1.1 Multi-Class Classification

One-of-many classification. Each sample can belong to ONE of C classes. The CNN will have C output neurons that can be gathered in a vector s (Scores). The target (ground truth) vector t will be a one-hot vector with a positive class and $C - 1$ negative classes.

This task is treated as a **single classification problem** of samples in one of C classes.

2.1.2 Multi-Label Classification

Each sample can belong to more than one class. The CNN will have as well C output neurons. The target vector t can have more than a positive class, so it will be a vector of 0s and 1s with C dimensionality.

This task is treated as C different binary $(C' = 2, t' = 0 \text{ } or \text{ } t' = 1)$ and independent classification problems, where each output neuron decides if a sample belongs to a class or not.(每一维度上我们都可以看成是一个二项分布，并且不同维度之间时相互独立的)

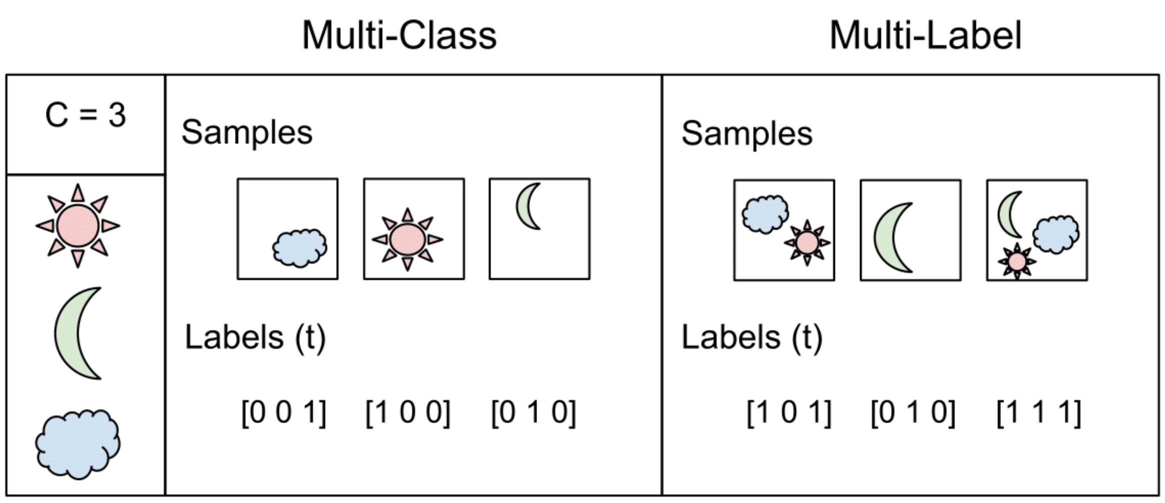


Fig.2 两种Task的实例

2.2 Output Activation Functions

These functions are transformations we apply to vectors coming out from CNNs (i.e. s) before the loss computation.

2.2.1 Sigmoid

It squashes(压缩) a vector in the range (0, 1). It is applied independently to each element of s . It's also called **logistic function**.(CNN的输出中的每一个数都会被Sigmoid函数单独作用)

The Sigmoid function can be computed as：

$$f(s_i) = \frac{1}{1 + e^{-s_i}} \tag{1}$$

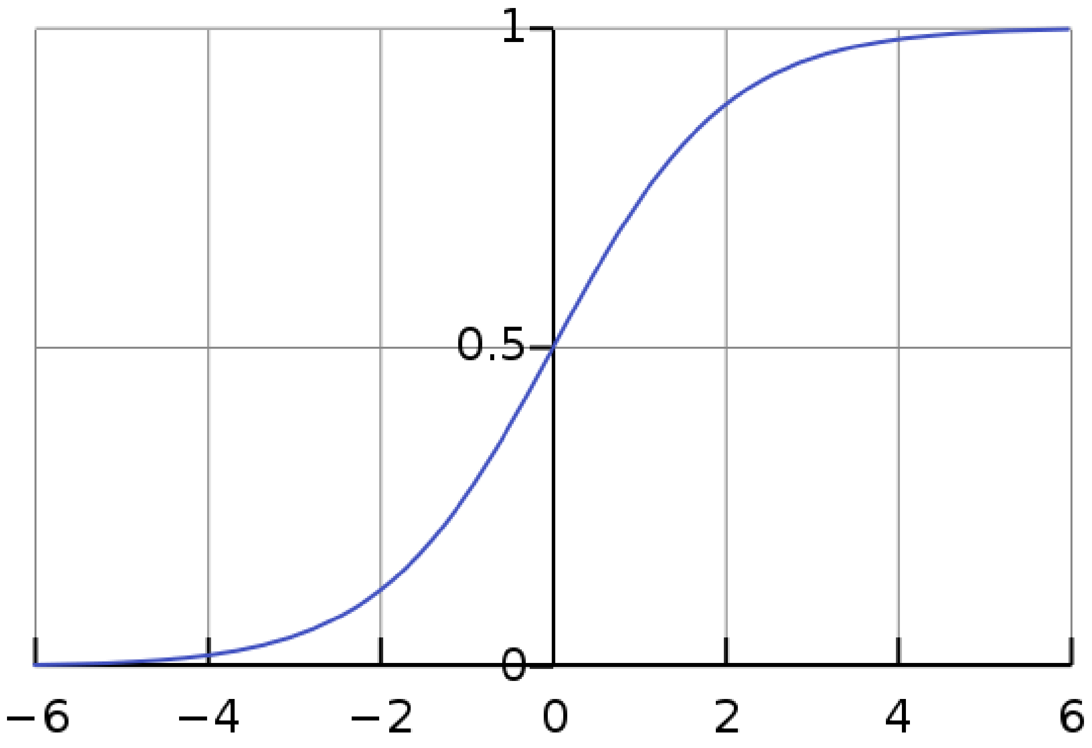


Fig.3 Sigmoid函数(或者说Logistic)的图像

2.2.2 Softmax

Softmax it's a function, not a loss. It squashes a vector in the range (0, 1) and all the resulting elements add up to 1. It is applied to the output scores s. As elements represent a class, **they can be interpreted as class probabilities**.

The Softmax function cannot be applied independently to each s_i, since it depends on all elements of s. For a given class s_i, the Softmax function can be computed as:

$$f(s)_i = \frac{e^{s_i}}{\sum_j^C e^{s_j}} \tag{2}$$

Where s_j are the scores inferred by the net for each class in C. Note that the Softmax activation for a class s_i depends on all the scores in s.

看了以上的说明，可以总结两个Softmax与Sigmoid函数的区别：

- Softmax是作用于整个输出s，而Sigmoid是单独作用于输出的每一个数s_i。因此我们可以看到式(1)中为$f(s_i)$，而式(2)为$f(s)_i$，两者接收的输入是不一样的。
- Softmax对计算后得到的$f(s)$有要求——所有维度上的数的和为1，而Sigmoid没这个要求。

我从一个网站,要科学上网上截取了更加详细的两者的区别：

	Softmax Function	Sigmoid Function
1	Used for multi-classification in logistic regression model.	Used for binary classification in logistic regression model.
2	The probabilities sum will be 1	The probabilities sum need not be 1.
3	Used in the different layers of neural networks.	Used as activation function while building neural networks.
4	The high value will have the higher probability than other values.	The high value will have the high probability but not the higher probability.

> *Activation functions are used to transform vectors before computing the loss in the training phase. In testing, when the loss is no longer applied, activation functions are also used to get the CNN outputs.*

Losses

3.1 Cross-Entropy loss

The **Cross-Entropy Loss** is actually the only loss we are discussing here. The other losses names written in the title are other names or variations of it. The CE Loss is defined as:

$$CE = -\sum_{i}^{C} t_i log(s_i) \tag{3}$$

Where t_i and s_i are the ground truth and the CNN score for each $class_i$ in C. As **usually an activation function (Sigmoid / Softmax) is applied to the scores before the CE Loss computation**, we write $f(s_i)$ to refer to the activations.

In a **binary classification problem**, where $C' = 2$, the Cross Entropy Loss can be defined also as:

$$CE = -\sum_{i=1}^{C'=2} t_i log(s_i) = -t_1 log(s_1) - (1 - t_1) log(1 - s_1) \tag{4}$$

Where it's assumed that there are two classes: C_1 and C_2. $t_1 \in [0, 1]$ and s_1 are the ground truth and the score for C_1, and $t_2 = 1 - t_1$ and $s_2 = 1 - s_1$ are the ground truth and the score for C_2.

> *That is the case when we split a Multi-Label classification problem in C binary classification problems. See next Binary Cross-Entropy Loss section for more details.*

Logistic Loss and **Multinomial Logistic Loss** are other names for **Cross-Entropy loss**.

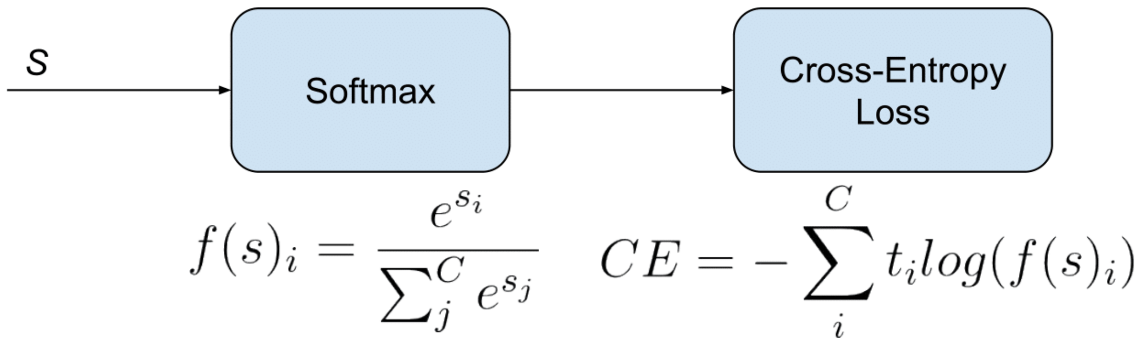
The layers of Caffe, Pytorch and Tensorflow than use a Cross-Entropy loss without an embedded activation function(不集成激活函数) are:

- Caffe: Multinomial Logistic Loss Layer. Is limited to multi-class classification (does not support multiple labels).

- Pytorch: BCELoss. Is limited to binary classification (between two classes).
- TensorFlow: log_loss.

3.2 Categorical Cross-Entropy loss

Also called **Softmax Loss**. It is a **Softmax activation** plus a **Cross-Entropy loss**. If we use this loss, we will train a CNN to output a probability over the C classes for each image. **It is used for multi-class classification.**



S
Softmax
Cross-Entropy Loss
$f(s)_i = \frac{e^{s_i}}{\sum_j^C e^{s_j}}$
$CE = -\sum_i^C t_i log(f(s)_i)$

Fig.4 Categorical Cross-Entropy loss计算流程图

In the specific (and usual) case of Multi-Class classification **the labels are one-hot**, so only the positive class C_p keeps its term in the loss. There is only one element of the Target vector t which is not zero $t_i = t_p$. So discarding the elements of the summation which are zero due to target labels, we can write:

$$CE = -log\left(\frac{e^{s_p}}{\sum_j^C e^{s_j}}\right) \tag{5}$$

Where s_p is the CNN score for the positive class.(式(5)已经把 Fig. 4 中的两个流程合并起来了)

Defined the loss, now we'll have to compute its **gradient respect to the output neurons** of the CNN in order to backpropagate it through the net and optimize the defined loss function tuning the net parameters. So we need to compute the gradient of CE Loss respect each CNN class score in s.

The loss terms coming from the negative classes are zero. However, the loss gradient respect those negative classes is not cancelled, since the Softmax of the positive class also depends on the negative classes scores.

The gradient expression will be the same for all C except for the ground truth class C_p, because the score of C_p (i.e. s_p) is in the numerator(分子).

After some calculus, the derivative with respect to the positive class is:

$$\frac{\partial}{\partial s_p}\left(-log\left(\frac{e^{s_p}}{\sum_j^C e^{s_j}}\right)\right) = \left(\frac{e^{s_p}}{\sum_j^C e^{s_j}} - 1\right) \tag{6}$$

And the derivative respect to the other (negative) classes is:

$$\frac{\partial}{\partial s_n}\left(-log\left(\frac{e^{s_p}}{\sum_j^C e^{s_j}}\right)\right) = \left(\frac{e^{s_n}}{\sum_j^C e^{s_j}}\right) \tag{7}$$

Where s_n is the score of any negative class in C different from C_p.

- Caffe: SoftmaxWithLoss Layer. Is limited to multi-class classification.
- Pytorch: CrossEntropyLoss. Is limited to multi-class classification, This criterion combines `LogSoftmax` and `NLLLoss` in one single class.(这里面集成了Softmax函数，会自动计算的，因此我们无需在此之前再调用Softmax)
- TensorFlow: softmax_cross_entropy. Is limited to multi-class classification.

> *In this Facebook work they claim that, despite being counter-intuitive,* **Categorical Cross-Entropy loss**, or **Softmax loss** *worked better than* **Binary Cross-Entropy loss** *in their multi-label classification problem.*

—> **Skip this part if you are not interested in Facebook or me using Softmax Loss for multi-label classification, which is not standard.**(Softmax也可以用于multi-label，但是并不是那么合理规范)

When Softmax loss is used is a multi-label scenario, the gradients get a bit more complex, since the loss contains an element for each positive class. Consider M are the positive classes of a sample. The CE Loss with Softmax activations would be:

$$CE = \frac{1}{M} \sum_{p}^{M} -log\left(\frac{e^{s_p}}{\sum_j^C e^{s_j}}\right)$$

Where each s_p in M is the CNN score for each positive class. As in Facebook paper, I introduce a scaling factor $1/M$ to make the loss invariant to the number of positive classes, which may be different per sample.

The gradient has different expressions for positive and negative classes. For positive classes:

$$\frac{\partial}{\partial s_{pi}}\left(\frac{1}{M} \sum_{p}^{M} -log\left(\frac{e^{s_p}}{\sum_j^C e^{s_j}}\right)\right) = \frac{1}{M}\left(\left(\frac{e^{s_{pi}}}{\sum_j^C e^{s_j}} - 1\right) + (M-1)\frac{e^{s_{pi}}}{\sum_j^C e^{s_j}}\right)$$

Where s_{pi} is the score of any positive class.

For negative classes:

$$\frac{\partial}{\partial s_n}\left(\frac{1}{M} \sum_{p}^{M} -log\left(\frac{e^{s_p}}{\sum_j^C e^{s_j}}\right)\right) = \frac{e^{s_n}}{\sum_j^C e^{s_j}}$$

This expressions are easily inferable from the single-label gradient expressions.

As Caffe Softmax with Loss layer nor Multinomial Logistic Loss Layer accept multi-label targets, I implemented my own PyCaffe Softmax loss layer, following the specifications of the Facebook paper. Caffe python layers let's us easily customize the operations done in the forward and backward passes of the layer:

- Forward pass: Loss computation

```
def forward(self, bottom, top):
   labels = bottom[1].data
   scores = bottom[0].data
   # Normalizing to avoid instability
   scores -= np.max(scores, axis=1, keepdims=True)
   # Compute Softmax activations
   exp_scores = np.exp(scores)
   probs = exp_scores / np.sum(exp_scores, axis=1, keepdims=True)
   logprobs = np.zeros([bottom[0].num,1])
   # Compute cross-entropy loss
   for r in range(bottom[0].num): # For each element in the batch
```

```
        scale_factor = 1 / float(np.count_nonzero(labels[r, :]))
        for c in range(len(labels[r,:])): # For each class 
            if labels[r,c] != 0:  # Positive classes
                logprobs[r] += -np.log(probs[r,c]) * labels[r,c] * scale_factor # We sum the loss per class for each element of the batch

    data_loss = np.sum(logprobs) / bottom[0].num

    self.diff[...] = probs  # Store softmax activations
    top[0].data[...] = data_loss # Store loss
```

We first compute Softmax activations for each class and store them in *probs*. Then we compute the loss for each image in the batch considering there might be more than one positive label. We use an *scale_factor* (M) and we also multiply losses by the labels, which can be binary or real numbers, so they can be used for instance to introduce class balancing. The batch loss will be the mean loss of the elements in the batch. We then save the *data_loss* to display it and the *probs* to use them in the backward pass.

- Backward pass: Gradients computation

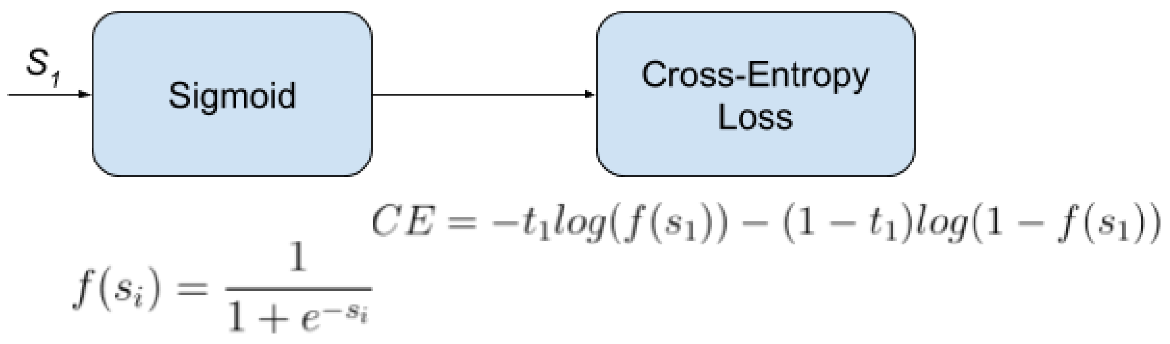
```
def backward(self, top, propagate_down, bottom):
   delta = self.diff   # If the class label is 0, the gradient is equal to probs
   labels = bottom[1].data
   for r in range(bottom[0].num):  # For each element in the batch
       scale_factor = 1 / float(np.count_nonzero(labels[r, :]))
       for c in range(len(labels[r,:])):  # For each class
           if labels[r, c] != 0:  # If positive class
               delta[r, c] = scale_factor * (delta[r, c] - 1) + (1 - scale_factor) * delta[r, c]
   bottom[0].diff[...] = delta / bottom[0].num
```

In the backward pass we need to compute the gradients of each element of the batch respect to each one of the classes scores s. As the gradient for all the classes C except positive classes MM is equal to *probs*, we assign *probs* values to *delta*. For the positive classes in M we subtract 1 to the corresponding *probs* value and use *scale_factor* to match the gradient expression. We compute the mean gradients of all the batch to run the backpropagation.

> *The Caffe Python layer of this Softmax loss supporting a multi-label setup with real numbers labels is available here*

3.3Binary Cross-Entropy Loss

Also called **Sigmoid Cross-Entropy loss**. It is a **Sigmoid activation** plus a **Cross-Entropy loss**. Unlike **Softmax loss**, it is independent for each vector component (class), meaning that the loss computed for every CNN output vector component is not affected by other component values. That’s why it is used for **multi-label classification**. The insight of an element belonging to a certain class should not influence the decision for another class.



s_1
Sigmoid
Cross-Entropy Loss
$CE = -t_1 log(f(s_1)) - (1 - t_1) log(1 - f(s_1))$
$f(s_i) = \frac{1}{1 + e^{-s_i}}$

Fig. 5 Binary Cross-Entropy Loss的计算流程

It’s called **Binary Cross-Entropy Loss** because it sets up a binary classification problem between $C' = 2$ classes for every class in C, as explained above. So when using this Loss, the formulation of **Cross Entropy Loss** for binary problems is often used:

$$CE = -\sum_{i=1}^{C'=2} t_i log(f(s_i)) = -t_1 log(f(s_1)) - (1 - t_1) log(1 - f(s_1)) \tag{8}$$

This would be the pipeline for each one of the C classes. We set C independent binary classification problems ($C' = 2$). Then we sum up the loss over the different binary problems: We sum up the gradients of every binary problem to backpropagate, and the losses to monitor the global loss.

And, in eq. (8), s_1 and t_1 are the score and the gorundtruth label for the class C'_1, which is also the class C_i in C. $s_2 = 1 - s_1$ and $t_2 = 1 - t_1$ are the score and the groundtruth label of the class C'_2, which is not a “class” in our original problem with C classes, but a class we create to set up the binary problem with $C_i = C_1$. We can understand it as a background class.

The loss can be expressed as:

$$CE = \begin{cases} -log(f(s_1)) & if \quad t_1 = 1 \\ -log(1 - f(s_1)) & if \quad t_1 = 0 \end{cases} \tag{9}$$

Where $t_1 = 1$ means that the class $C_i = C'_1$ is positive for this sample.

In this case, the activation function does not depend in scores of other classes in C more than $C_i = C'_1$. So the gradient respect to the each score s_i in s will only depend on the loss given by its binary problem.

The gradient respect to the score $s_i = s_1$ can be written as:

$$\frac{\partial}{\partial s_i}(CE(f(s_i))) = t_1(f(s_1) - 1)) + (1 - t_1) f(s_1) \tag{10}$$

Where $f()$ is the **Sigmoid** function. It can also be written as:

$$\frac{\partial}{\partial s_i}(CE(f(s_i)) = \begin{cases} f(s_i) - 1 & if \quad t_i = 1 \\ f(s_i) & if \quad t_i = 0 \end{cases}$$

原文中没有对式(10)的推导过程，我尝试推了一下，其实很简单，主要是注意**Sigmoid**函数$f(x)$的导数具有如下性质：

$$\frac{\mathrm{d}f}{\mathrm{d}x} = f \cdot (1 - f)$$

> *Refer here for a detailed loss derivation.*

- Caffe: Sigmoid Cross-Entropy Loss Layer

- Pytorch: BCELoss, creates a criterion that measures the Binary Cross Entropy between the target and the output. **Note that the targets should be numbers between 0 and 1**.
- TensorFlow: sigmoid_cross_entropy.