# 目录



# Softmax 函数

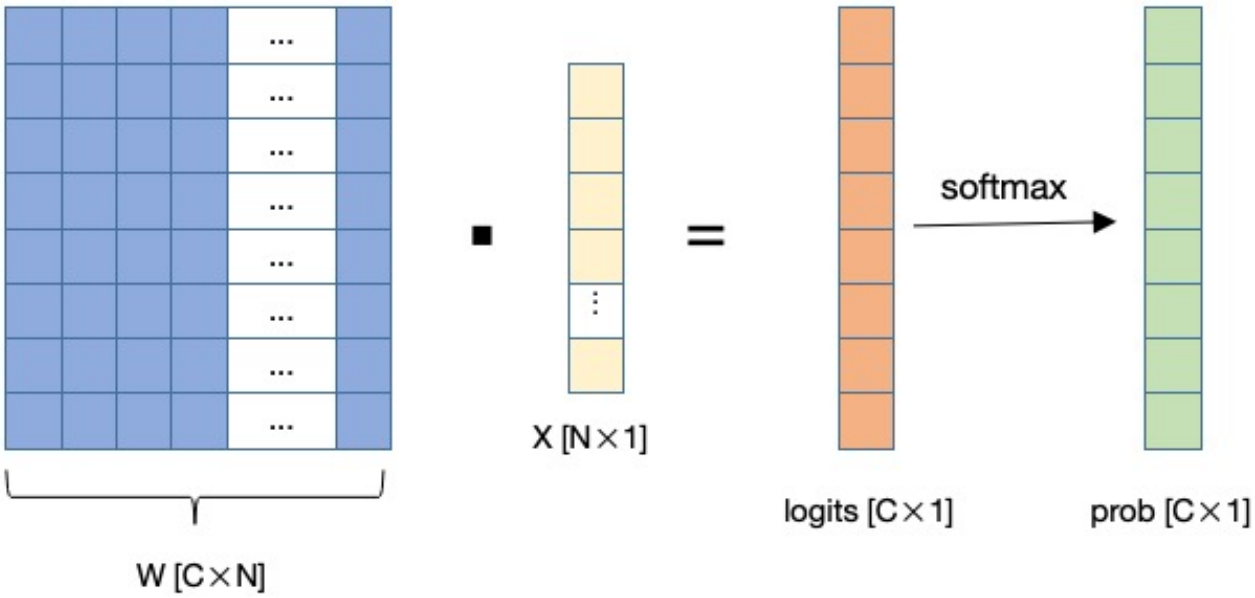
【参考】

- wiki - Softmax函数
- csdn - 卷积神经网络系列之softmax，softmax loss和cross entropy的讲解

在数学，尤其是概率论和相关领域中，Softmax函数，或称**归一化指数函数**，是**逻辑函数**的一种推广。它能将一个含任意实数的K维向量 $\mathbf{z}$ “压缩” 到另一个K维实向量 $\sigma(\mathbf{z})$ 中，使得每一个元素的范围都在 $(0,1)$ 之间，并且所有元素的和为1。

Softmax函数实际上是有限项离散概率分布的梯度对数归一化。因此，Softmax函数在包括 多项逻辑回归，多项线性判别分析，朴素贝叶斯分类器和人工神经网络等的多种基于概率的多分类问题方法中都有着广泛应用。

如在神经网络中，用在全连接层进行分类：如下图：




这张图的等号左边部分就是全连接层做的事，W是全连接层的参数，我们也称为权值，X是全连接层的输入，也就是特征。

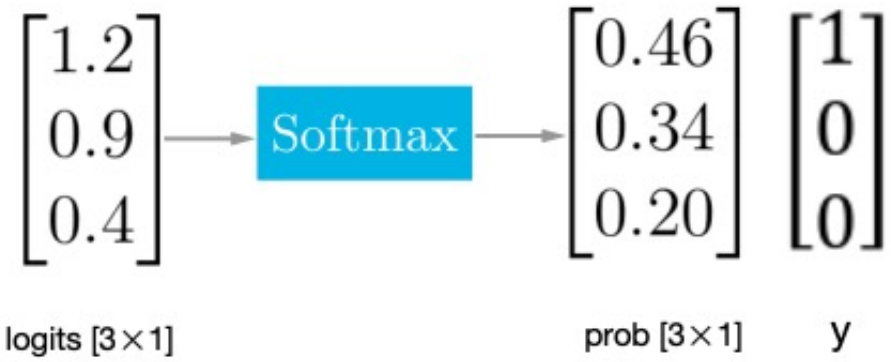
从图上可以看出特征X是N×1的向量，这是怎么得到的呢？这个特征向量就是由全连接层前面多个卷积层和池化层处理后得到的，假设全连接层前面连接的是一个卷积层，这个卷积层的输出是100个（channel）feature map，每个 feature map 的大小（W×H）是4×4，那么在将这些特征输入给全连接层之前会将这些特征展平成N×1的向量（这个时候N就是100×4×4=1600）。

W是全连接层的参数，是个 C×N 的矩阵，这个N和X是N对应的，C表示类别数，比如你是7分类，那么 C 就是7。我们所说的训练一个网络，对于全连接层而言就是寻找最合适的W矩阵。因此全连接层就是执行 W·X 得到一个 C×1 的向量（也就是图中的logits[C×1]），这个向量里面的每个数都没有大小限制的，也就是从负无穷大到正无穷大。

然后如果你是多分类问题，一般会在全连接层后面接一个softmax层，这个softmax的输入是 C×1 的向量，输出也是 C×1 的向量（也就是图中的prob[C×1]，这个向量的每个值表示这个样本属于每个类的概率），只不过输出的向量的每个值的大小范围为0到1，和为1。在神经网络中 Softmax 的计算公式如下：

$$a_i = \frac{e^{z_i}}{\sum_{k=1}^{C} e^{z_k}} \tag{1}$$

其中 T 表示类别的数目，就是上图中的 logits[T×1]，$z_i$，$z_k$ 表示其中的一个类别的值，$a_i$ 表示是某一个类别的概率。这样就能保证 $\sum_{j=1}^{C} a_j = 1$。如下图：




# Softmax 损失函数

【参考】

- 简书 - 简单易懂的softmax交叉熵损失函数求导
- 知乎 - 详解softmax函数以及相关求导过程
- github.io - 彻底理解softmax

在神经网络反向传播中，要求一个损失函数，这个损失函数其实表示的是真实值与网络的估计值的误差，知道误差了，才能知道怎样去修改网络中的权重。

Softmax 的损失函数如下：

$$l = -\sum_{i=1}^{C} y_i \ln a_i \tag{2}$$

其中：

- C 表示样本的输出类别数，如上图输出三个值，因此 C=3；
- $y_i$ 表示样本的标签向量的第 i 个值，他的大小为C×1，但其中只有一个是1表示样本的真实标签，其他的都是零，如上图的 y；
- $a_i$ 表示 Softmax 预测第 i 个样本类别概率，如上图的prob[3×1]。

从上面的第二条可以看到，只有样本对应的真实标签的 a 才起作用，上面的损失函数又可以简化为：

$$l = -\ln a_i \tag{3}$$

其中的 i 表示真实的标签，如上图可以知道这时的 i 是 1，因为 1 在首位，那么对应的$a_1$就是0.46，所以损失就是$-ln(0.46) = 0.7765$。如果标签向量是 y=[0,1,0]，那么此时的 i 为 2 ，对应的 $a_2$就是0.34，所以损失就是$-ln(0.34) = 1.0788$。可以看到损失提高了很多。

# 交叉熵

其实 (2) 式就已经是交叉熵损失函数，只不过计算的是一个样本的损失，如果我们有 m 个样本，就需要计算 m 个损失：

$$L = -\sum_{k=1}^{m}\sum_{i=1}^{C} y_{ki} \ln a_{ki} \tag{4}$$

$y_{ki}$ 是第 k 个样本第 i 个输出的标签，如第2个样本标签向量为 [0,1,0,0]，那么$y_{22} = 1$。$a_{ki}$ 是第 k 个样本第 i 个Softmax输出。

# 求导

有了交叉熵函数，就可以对其求导，然后找到最优解。此处以一个样本为例，即使用公式（2），对其求导。

需要明确的是，我们是要对公式（1）中的 z 求导，只有这样我们才能找到参数的最优解，我们求导的是一个复合函数：

$$\frac{\partial l}{\partial z_i}$$

其中，

$$l = -\sum_{i=1}^{C} y_i \ln a_i \ ,$$

$$a_i = \frac{e^{z_i}}{\sum_{k=1}^{C} e^{z_k}} = \frac{e^{z_i}}{e^{z_1} + \ldots + e^{z_i} + \ldots + e^{z_C}}$$

根据链式求导法则：

$$\frac{\partial l}{\partial z_i} = \frac{\partial l}{\partial a_j}\frac{\partial a_j}{\partial z_i} \tag{5}$$

有个人可能有疑问了，这里为什么是 $a_j$ 而不是 $a_i$，这里要看一下softmax的公式了，因为softmax公式的特性，它的分母包含了所有神经元的输出，所以，对于不等于 i 的其他输出里面，也包含着 $z_i$ ，所有的a都要纳入到计算范围中，并且后面的计算可以看到需要分为i = j和i ≠ j两种情况求导。

# 第一部分求导

对于（5）式的第一部分求导有：

$$\frac{\partial l}{\partial a_j} = \frac{\partial \left(-\sum_{j=1}^{C} y_j \ln a_j\right)}{\partial a_j} = -\sum_{j=1}^{C} y_j \frac{1}{a_j} \tag{6}$$

# 第二部分求导

第二部分求导稍微复杂一点，需要分为两种情况:

① 首先是$i = j$，这里会用到如下的求导公式：

$$\left(\frac{u}{v}\right)' = \frac{u'v - uv'}{v^2}$$

因此：

$$\begin{aligned} \frac{\partial a_j}{\partial z_i} &= \frac{\partial \left(\frac{e^{z_i}}{\sum_{k=1}^{C} e^{z_k}}\right)}{\partial z_i} = \frac{e^{z_i}\sum_{k=1}^{C} e^{z_k} - \left(e^{z_i}\right)^2}{\left(\sum_{k=1}^{C} e^{z_k}\right)^2} \\ &= \frac{e^{z_i}}{\sum_{k=1}^{C} e^{z_k}}\left(1 - \frac{e^{z_i}}{\sum_{k=1}^{C} e^{z_k}}\right) \\ &= a_i(1 - a_i) \end{aligned} \tag{7}$$

② 当 $i \neq j$ 时：

$$\begin{aligned} \frac{\partial a_j}{\partial z_i} &= \frac{\partial \left(\frac{e^{z_j}}{\sum_{k=1}^{C} e^{z_k}}\right)}{\partial z_i} = -e^{z_j}\left(\frac{1}{\sum_{k=1}^{C} e^{z_k}}\right)^2 e^{z_i} \\ &= -\frac{e^{z_j}}{\sum_{k=1}^{C} e^{z_k}}\frac{e^{z_i}}{\sum_{k=1}^{C} e^{z_k}} \\ &= -a_j a_i \end{aligned} \tag{8}$$

综合 （6） （7） （8）就有：

$$
\begin{aligned}
\frac{\partial l}{\partial z_i} &= \left( -\sum_{j=1}^{C} y_j \frac{1}{a_j} \right) \frac{\partial a_j}{\partial z_i} \\
&= -\frac{y_i}{a_i} a_i (1 - a_i) + \sum_{i \neq j} \frac{y_j}{a_j} a_i a_j \\
&= -y_i + a_i \sum_j y_j
\end{aligned} \tag{9}
$$

## 最终结果

可以看到（9）式已经非常简洁了。对于分类问题，我们最终的结果$y_i$最终只有一个类别是 1 ，其他类别都是 0 ，因此对于分类问题，这个梯度就等于：

$$
\frac{\partial l}{\partial z_i} = a_i - y_i
$$

# Softmax 值的稳定性

【参考】

- csdn - Softmax函数与交叉熵

在Python中，softmax函数为：

```
def softmax(x):
    exp_x = np.exp(x)
    return exp_x / np.sum(exp_x)
```

传入较小的数时，还可以正常的计算，但是当传入较大的值时，会出现 nan 的情况。这是因为在求`exp(x)`时候溢出了。

一种简单有效避免该问题的方法就是让exp(x)中的x值不要那么大或那么小，在softmax函数的分式上下分别乘以一个非零常数：

$$
a_i = \frac{e^{z_i}}{\sum_{k=1}^{C} e^{z_k}} = \frac{Ee^{z_i}}{\sum_{k=1}^{C} Ee^{z_k}} = \frac{e^{z_i + log(E)}}{\sum_{k=1}^{C} e^{z_k + log(E)}} = \frac{e^{z_i + F}}{\sum_{k=1}^{C} e^{z_k + F}}
$$

这里log(E)是个常数，所以可以令它等于 F。加上常数 F 之后，等式与原来还是相等的，所以我们可以考虑怎么选取常数 F。我们的想法是让所有的输入在0附近，这样 $e^{z_i}$ 的值不会太大，所以可以让 F 的值为：

$$
F = -max(z_1, z_2, \ldots, z_C)
$$

这样子将所有的输入平移到0附近（当然需要假设所有输入之间的数值上较为接近），同时，除了最大值，其他输入值都被平移成负数，e为底的指数函数，越小越接近0，这种方式比得到 nan 的结果更好。

这样新的函数就为：

```
def softmax(x):
    shift_x = x - np.max(x)
    exp_x = np.exp(shift_x)
    return exp_x / np.sum(exp_x)
```