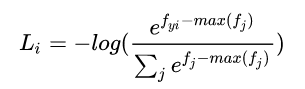
在 考虑数值计算稳定性情况下的Softmax损失函数的公式如下 ：



对所有样本及计入正则化惩罚后，损失函数公式为：



我们先从 Li看起。

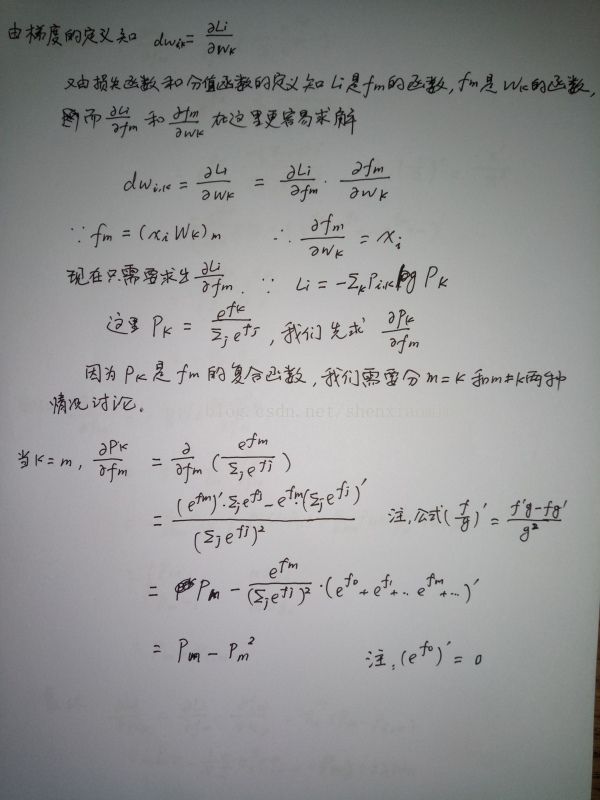
f(i,j)即矩阵f(x,w)中的第i,j个元素。我们与之前一样求出样本集与权重集之间的评分集即可。

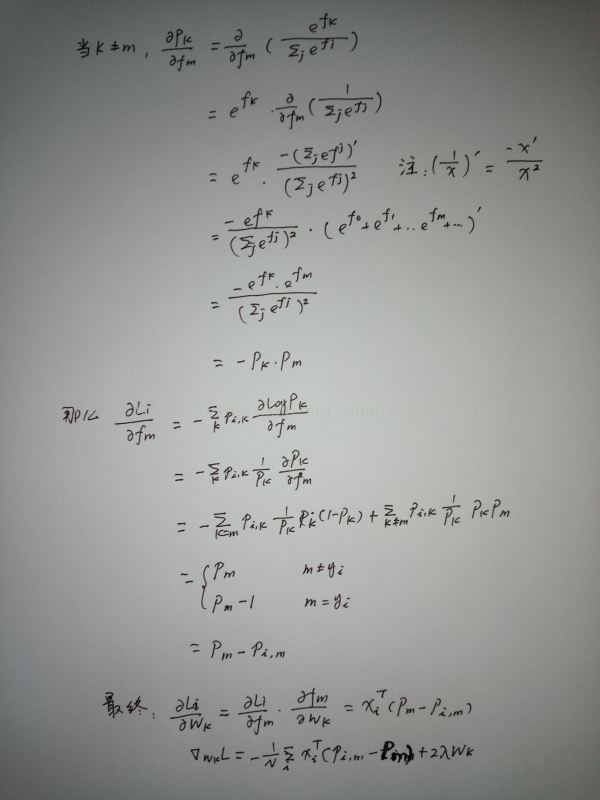
max(fj)即在第i个样本的所有分类得分中最大的得分。从式中看，评分集中的每一个元素都需要减去这个最大得分，这可以通过矩阵操作的广播机制来完成。同时，广播机制对指数运算也一样有效。因此损失函数可以计算为：

1. f **=** X**.**dot(W) *# N by C*
2. *# f\_max是对第i行元素的所有分类计分求最大值，所以axis = 1*
3. f\_max **=** np**.**reshape(np**.**max(f, axis**=**1), (num\_train, 1)) *# N by 1*
4. *# 对每个计分求归一化概率，这个概率是每个样本在不同类别上的分布。 N by C*
5. prob **=** np**.**exp(f **-** f\_max) **/** np**.**sum(np**.**exp(f **-** f\_max), axis**=**1, keepdims**=**True)
7. **for** i **in** xrange(num\_train):
8. **for** j **in** xrange(num\_class):
9. **if** (j **==** y[i]):
10. loss **+=** **-**np**.**log(prob[i, j]))
12. loss **/=** num\_train
13. loss **+=** 0.5 **\*** reg **\*** np**.**sum(W **\*** W)

这个损失函数计算的原理是，如果对于样本i，它的正确分类类别是j，那么如果 prob[i,j]的值为1，则说明分类正确，这种情况下对损失函数没有贡献。而如果分类错误，则prob[i,j]的值将是一个小于1的值，这种情况下将对损失函数有所贡献。优化权重将有可能使得prob[i,j]趋近于1，从而损失函数最小。在未经训练时，由于权重是随机生成的，因此应该每个分类的概率就是10%，因此 loss应该接近 -log (0.1) （在没有加正则化惩罚的情况下）.

梯度的推导如下：





上图中， （注意这里字母p是小写）是样本的分类概率，是一个Cx1的向量（假设有C个分类的话）。当m为正确分类时，其值为1，其它元素取值为0。这里Pm是即P[i,m]，是样本i在第m个分类上的概率。 在求损失函数时，我们已经得到了概率矩阵了，所以P[i,m]已知。

简书网友Deeplayer有一个类似的[推导](http://link.zhihu.com/?target=http%3A//upload-images.jianshu.io/upload_images/2301760-1c7b8c12bbe6a1bc.png%3FimageMogr2/auto-orient/strip%257CimageView2/2/w/1240" \t "_blank)，省去了一些中间过程，更为简洁和清晰。

加上梯度后的softmax\_loss\_naive版本：

1. num\_train **=** X**.**shape[0]
2. num\_classes **=** W**.**shape[1]
3. f **=** X**.**dot(W) *#N by C*
4. f\_max **=** np**.**reshape(np**.**max(f, axis **=** 1), (num\_train, 1))
5. prob **=** np**.**exp(f **-** f\_max)**/**np**.**sum(np**.**exp(f**-**f\_max), axis **=** 1, keepdims **=** True)
7. **for** i **in** xrange(num\_train):
8. **for** j **in** xrange(num\_classes):
9. **if** (j **==** y[i]):
10. loss **+=** **-**np**.**log(prob[i,j])
11. dW[:,j] **+=** (1 **-** prob[i,j]) **\*** X[i]
12. **else**:
13. dW[:,j] **-=** prob[i,j] **\*** X[i]
15. loss **/=** num\_train
16. loss **+=** 0.5 **\*** reg **\*** np**.**sum(W**\***W)
17. dW **=** **-**dW **/** num\_train **+** reg **\*** W

从 softmax\_loss\_naive出发，看看如何去掉循环：

loss出现在内循环中，只不过只有当 j==y[i]时，我们才把 -np.log(prob[i,j])加上去。所以如果我们能找出 j!=y[i] 的那些元素，将其prob[i,j]置为1，从而np.log(prob[i,j]) == 0，这样就可以直接对矩阵求和了。在linear\_svm.py中，我们使用margins[np.arange(num\_train), y] = 0的方法来对少数元素的值做更新，但这里我们需要的条件是j!= y[i]，而非j==y[i]，所以还要再想别的办法。对dW矩阵而言，应该是先在prob的基础上作出一个新的矩阵，使其元素为 prob的对应元素的负数，然后将那些 j==y[i]的元素加上1，然后将这个新的矩阵与X相乘。这两个矩阵刚好都可以用下面的keepProb来实现。

下面的代码是 softmax.py中TODO：

1. *#################################################################*
2. *# TODO: Compute the softmax loss and its gradient using no explicit loops. #*
3. *# Store the loss in loss and the gradient in dW. If you are not careful #*
4. *# here, it is easy to run into numeric instability. Don't forget the #*
5. *# regularization! #*
6. *################################################################*

的内容：

1. num\_train **=** X**.**shape[0]
2. num\_classes **=** W**.**shape[1]
3. f **=** X**.**dot(W) *#N by C*
4. f\_max **=** np**.**reshape(np**.**max(f, axis **=** 1), (num\_train, 1))
5. prob **=** np**.**exp(f **-** f\_max)**/**np**.**sum(np**.**exp(f**-**f\_max), axis **=** 1, keepdims **=** True)
7. keepProb **=** np**.**zeros\_like(prob)
8. keepProb[np**.**arange(num\_train), y] **=** 1.0
9. loss **+=** **-**np**.**sum(keepProb **\*** np**.**log(prob)) **/** num\_train **+** 0.5 **\*** reg **\*** np**.**sum(W**\***W)
10. dW **+=** **-**np**.**dot(X**.**T, keepProb **-** prob)**/**num\_train **+** reg **\*** W