The Cross-Entropy Loss Function for the Softmax Function

作者：凯鲁嘎吉 - 博客园 http://www.cnblogs.com/kailugaji/

本文介绍含有softmax函数的交叉熵损失函数的求导过程，并介绍一种交叉熵损失的等价形式，从而解决因log(0)而出现数值为NaN的问题。

1. softmax函数求导

Softmax classification with cross-entropy

➢Derivative of the softmax function

$$Q:N\times K,\ P:N\times K,\ a:N\times K$$

$$Q=[\boldsymbol{q}_i]_{N\times 1}=[q_{ik}]_{N\times K}\text{为常数，}P=[\boldsymbol{p}_i]_{N\times 1}=[p_{ik}]_{N\times K}$$

$$p_{ik}=\mathrm{softmax}(a_{ik})=\frac{e^{a_{ik}}}{\sum_{k=1}^{K}e^{a_{ik}}}=\frac{e^{a_{ik}}}{e^{a_{ik}}+\sum_{j\neq k}^{K}e^{a_{ij}}}$$

$$\text{由于}Q\text{与}P\text{都是概率分布，因此}\sum_{j=1}^{K}p_{ij}=\sum_{j=1}^{K}q_{ij}=1$$

$$\text{当}j=k\text{，}\frac{\partial p_{ik}}{\partial a_{ik}}=\frac{e^{a_{ik}}\sum_{k=1}^{K}e^{a_{ik}}-(e^{a_{ik}})^2}{\left(\sum_{k=1}^{K}e^{a_{ik}}\right)^2}=p_{ik}(1-p_{ik})$$

$$\text{当}j\neq k\text{，}\frac{\partial p_{ik}}{\partial a_{ij}}=\frac{-e^{a_{ik}}e^{a_{ij}}}{\left(\sum_{k=1}^{K}e^{a_{ik}}\right)^2}=-p_{ik}p_{ij}$$

1

2. 交叉熵损失求导(含softmax)

Softmax classification with cross-entropy

➢Derivative of the cross-entropy loss function for the softmax function

$$L_1=-\sum_{i=1}^{N}\sum_{k=1}^{K}q_{ik}\log p_{ik}$$

$$\begin{aligned}\frac{\partial L_1}{\partial \boldsymbol{a}_i}&=\frac{\partial\left(-\sum\limits_{k=1}^{K}q_{ik}\log p_{ik}\right)}{\partial \boldsymbol{a}_i}=-\sum_{k=1}^{K}q_{ik}\frac{\partial \log p_{ik}}{\partial \boldsymbol{a}_i}=-\sum_{k=1}^{K}\frac{q_{ik}}{p_{ik}}\frac{\partial p_{ik}}{\partial \boldsymbol{a}_i}\\&=-\frac{\boldsymbol{q}_i}{\boldsymbol{p}_i}\frac{\partial \boldsymbol{p}_i}{\partial \boldsymbol{a}_i}-\sum_{j\neq k}^{K}\frac{q_{ij}}{p_{ij}}\frac{\partial p_{ij}}{\partial a_{ik}}=-\frac{\boldsymbol{q}_i}{\boldsymbol{p}_i}\boldsymbol{p}_i(1-\boldsymbol{p}_i)-\sum_{j\neq k}^{K}\frac{q_{ij}}{p_{ij}}\left(-p_{ij}p_{ik}\right)\\&=-\boldsymbol{q}_i+\boldsymbol{p}_i\boldsymbol{q}_i+\boldsymbol{p}_i\sum_{j\neq k}^{K}q_{ij}=-\boldsymbol{q}_i+\boldsymbol{p}_i\sum_{j=1}^{K}q_{ij}=\boldsymbol{p}_i-\boldsymbol{q}_i\end{aligned}$$

2

3. 交叉熵损失函数(含softmax函数)的等价形式

Softmax classification with cross-entropy

➢The equivalent form of the cross-entropy loss function for the softmax function

$$\begin{aligned}\nabla_{\boldsymbol{a}_i}L_1&=\nabla_{\boldsymbol{a}_i}\left(-\sum_{k=1}^{K}q_{ik}\log p_{ik}\right)=\nabla_{\boldsymbol{a}_i}\left(-\sum_{k=1}^{K}q_{ik}\log p_{ik}\right)+\nabla_{\boldsymbol{a}_i}\sum_{k=1}^{K}p_{ik}=\nabla_{\boldsymbol{a}_i}\left(-\sum_{k=1}^{K}q_{ik}\log p_{ik}\right)+\sum_{k=1}^{K}\nabla_{\boldsymbol{a}_i}p_{ik}\\&=\nabla_{\boldsymbol{a}_i}\left(-\sum_{k=1}^{K}q_{ik}\log p_{ik}\right)+\sum_{k=1}^{K}p_{ik}\frac{\nabla_{\boldsymbol{a}_i}p_{ik}}{p_{ik}}=\nabla_{\boldsymbol{a}_i}\left(-\sum_{k=1}^{K}q_{ik}\log p_{ik}\right)+\sum_{k=1}^{K}p_{ik}\nabla_{\boldsymbol{a}_i}\log p_{ik}\\&=\nabla_{\boldsymbol{a}_i}\left(-\sum_{k=1}^{K}q_{ik}\log\frac{e^{a_{ik}}}{\sum_{k=1}^{K}e^{a_{ik}}}\right)+\sum_{k=1}^{K}p_{ik}\nabla_{\boldsymbol{a}_i}\log\frac{e^{a_{ik}}}{\sum_{k=1}^{K}e^{a_{ik}}}\\&=-\sum_{k=1}^{K}q_{ik}\nabla_{\boldsymbol{a}_i}\left(a_{ik}-\log\left(\sum_{k=1}^{K}e^{a_{ik}}\right)\right)+\sum_{k=1}^{K}p_{ik}\nabla_{\boldsymbol{a}_i}\left(a_{ik}-\log\left(\sum_{k=1}^{K}e^{a_{ik}}\right)\right)\\&=\nabla_{\boldsymbol{a}_i}\sum_{k=1}^{K}(p_{ik}-q_{ik})a_{ik}+\left(\sum_{k=1}^{K}q_{ik}-\sum_{k=1}^{K}p_{ik}\right)\nabla_{\boldsymbol{a}_i}\log\left(\sum_{k=1}^{K}e^{a_{ik}}\right)=\nabla_{\boldsymbol{a}_i}\sum_{k=1}^{K}(p_{ik}-q_{ik})a_{ik}\end{aligned}$$

3

Softmax classification with cross-entropy

➢The equivalent form of the cross-entropy loss function for the softmax function

$$L_2=\sum_{i=1}^{N}\sum_{k=1}^{K}(p_{ik}-q_{ik})a_{ik}$$

$$\begin{aligned}\frac{\partial L_2}{\partial \boldsymbol{a}_i}&=\frac{\partial\left(\sum\limits_{k=1}^{K}(p_{ik}-q_{ik})a_{ik}\right)}{\partial \boldsymbol{a}_i}=\sum_{k=1}^{K}\frac{\partial\left((p_{ik}-q_{ik})a_{ik}\right)}{\partial \boldsymbol{a}_i}=\frac{\partial\left((p_{ik}-q_{ik})a_{ik}\right)}{\partial a_{ik}}+\sum_{j\neq k}^{K}\frac{\partial\left((p_{ij}-q_{ij})a_{ij}\right)}{\partial a_{ik}}\\&=\frac{\partial p_{ik}}{\partial a_{ik}}\boldsymbol{a}_i+(\boldsymbol{p}_i-\boldsymbol{q}_i)+\sum_{j\neq k}^{K}\frac{\partial p_{ij}}{\partial a_{ik}}a_{ij}=\boldsymbol{p}_i(1-\boldsymbol{p}_i)\boldsymbol{a}_i+(\boldsymbol{p}_i-\boldsymbol{q}_i)-\boldsymbol{p}_i\sum_{j\neq k}^{K}p_{ij}a_{ij}\\&=\boldsymbol{p}_i-\boldsymbol{q}_i+\boldsymbol{p}_i\boldsymbol{a}_i-\boldsymbol{p}_i\sum_{j=1}^{K}p_{ij}a_{ij}=\boldsymbol{p}_i-\boldsymbol{q}_i+\boldsymbol{p}_i\left(\boldsymbol{a}_i-\sum_{j=1}^{K}p_{ij}a_{ij}\right)\\&=\boldsymbol{p}_i-\boldsymbol{q}_i+\sum_{j=1}^{K}p_{ij}a_{ij}-\sum_{j=1}^{K}p_{ij}a_{ij}=\boldsymbol{p}_i-\boldsymbol{q}_i\end{aligned}$$

用到$\mathbb{E}(X-\mathbb{E}X)=0$

4

4. Python验证${p_i}\left({{a_i} - \sum\limits_{j = 1}^K {{p_{ij}}{a_{ij}}} } \right) = 0$

```python
# -*- coding: utf-8 -*-
# Author：凯鲁嘎吉 Coral Gajic
# https://www.cnblogs.com/kailugaji/
# 验证E(X-EX)=0
import numpy as np
a = np.random.rand(5)
p = np.random.rand(5)
p = p / p.sum(axis=0, keepdims=True)
b = (np.dot(p, a)).sum()
print('a: ', a)
print('p: ', p)
print('b: ', b)
print('结果: ', (p * (a - b)).sum())
```

结果:

```
D:\ProgramData\Anaconda3\python.exe "D:/Python code/2023.3 exercise/向量间的距离度量/test.py"
a:  [0.90457897 0.08975555 0.6955031  0.74161145 0.50095907]
p:  [0.02797057 0.09509987 0.27454503 0.273575   0.32880953]
b:  0.5923907183986392
结果:  5.204170427930421e-17

Process finished with exit code 0
```

5. Python验证L_1与L_2等价

```python
# -*- coding: utf-8 -*-
# Author：凯鲁嘎吉 Coral Gajic
# https://www.cnblogs.com/kailugaji/
# Softmax classification with cross-entropy
import torch
import numpy as np
import matplotlib.pyplot as plt
plt.rc('font',family='Times New Roman')

def sinkhorn(scores, eps = 5.0, n_iter = 3):
    def remove_infs(x):
        mm = x[torch.isfinite(x)].max().item()
        x[torch.isinf(x)] = mm
        x[x==0] = 1e-38
        return x
    scores = torch.tensor(scores)
    n, m = scores.shape
    scores1 = scores.view(n*m)
    Q = torch.softmax(-scores1/eps, dim=0)
    Q = remove_infs(Q).view(n,m).T
    r, c = torch.ones(n), torch.ones(m) * (n / m)
    for _ in range(n_iter):
        u = (c/torch.sum(Q, dim=1))
        Q *= remove_infs(u).unsqueeze(1)
        v = (r/torch.sum(Q,dim=0))
        Q *= remove_infs(v).unsqueeze(0)
    bsum = torch.sum(Q, dim=0, keepdim=True)
    Q = Q / remove_infs(bsum)
    P = Q.T
    assert torch.isnan(P.sum())==False
    return P

n = 128
m = 64
loss_1 = []
loss_2 = []
for _ in range(20):
    a = torch.rand(n, m)
    a = a ** 4
    a = a / a.sum(dim = -1, keepdim = True)
    P = torch.softmax(a, dim=1)
    b = np.random.rand(n, m)
    b = b ** 1.5
    Q = sinkhorn(b, 0.5, 10)
    # 方法1：
    loss_11 = -(Q * torch.log(P)).sum(-1).mean()
    loss_1.append(loss_11)
    # 方法2：
    loss_22 = ((P - Q) * a).sum(-1).mean()
    loss_2.append(loss_22)

loss_1, index = torch.sort(torch.tensor(loss_1), 0)
loss_2 = np.array(loss_2)[index]
print('方法1--损失：\n', np.array(loss_1))
print('方法2--损失：\n', loss_2)
grad_1 = np.gradient(np.array(loss_1))
grad_2 = np.gradient(np.array(loss_2))
print('方法1--梯度：\n', np.array(grad_1))
print('方法2--梯度：\n', np.array(grad_2))
plt.subplots(1, 2, figsize=(16, 7))
```

```
plt.subplot(1, 2, 1)
plt.plot(loss_1, loss_2, color = 'red', ls = '-')
plt.xlabel('Loss 1')
plt.ylabel('Loss 2')
plt.subplot(1, 2, 2)
plt.scatter(grad_1*1E6, grad_2*1E6, color = 'blue')
plt.xlabel('Gradient 1')
plt.ylabel('Gradient 2')
plt.savefig('softmax cross-entropy loss.png', bbox_inches='tight', dpi=500)
plt.show()
```

结果:

```
D:\ProgramData\Anaconda3\python.exe "D:/Python code/2023.3 exercise/向量间的距离度量/softmax_cross_entropy_loss_test.py"
方法1--损失:
 [4.15883989 4.15890663 4.15894403 4.15897117 4.15902341 4.15904347
 4.1590823  4.1590913  4.15910622 4.15913114 4.15913474 4.1591434
 4.15914856 4.15916808 4.15916826 4.15917904 4.15918445 4.15918608
 4.15925961 4.15926385]
方法2--损失:
 [0.00017298 0.00024753 0.00028277 0.00030974 0.00036783 0.0003804
 0.00041808 0.00043415 0.00044729 0.00047444 0.00047943 0.00048301
 0.00049451 0.00050864 0.00051069 0.0005161  0.00053111 0.00052533
 0.00060765 0.0006024 ]
方法1--梯度:
 [6.67441917e-05 5.20709542e-05 3.22701985e-05 3.96937794e-05
 3.61504422e-05 2.94408723e-05 2.39134622e-05 1.19608872e-05
 1.99222036e-05 1.42617498e-05 6.12662792e-06 6.90728703e-06
 1.23429982e-05 9.85375545e-06 5.47883732e-06 8.09470613e-06
 3.52000565e-06 3.75770611e-05 3.88866185e-05 4.24487449e-06]
方法2--梯度:
 [ 7.45563606e-05  5.48997609e-05  3.11016626e-05  4.25261140e-05
  3.53301332e-05  2.51239797e-05  2.68772106e-05  1.46074152e-05
  2.01447210e-05  1.60698704e-05  4.28303591e-06  7.54056029e-06
  1.28157065e-05  8.08914041e-06  3.73246714e-06  1.02123578e-05
  4.61507539e-06  3.82697805e-05  3.85335993e-05 -5.25437451e-06]

Process finished with exit code 0
```

可以看到，虽然理论上可以证明L_1与L_2的梯度相等，但实际应用时还是有点偏颇，当数据量与数据维度足够大时，L_1与L_2的数值呈现线性相关。

6. 参考文献

[1] Softmax classification with cross-entropy (2/2)

[2] Liu Q, Zhou Q, Yang R, et al. Robust Representation Learning by Clustering with Bisimulation Metrics for Visual Reinforcement Learning with Distractions[C]. AAAI, 2023.

[3] Python小练习：Sinkhorn-Knopp算法