date: 20230331
aim: 手动计算全连接网络，反向传播迭代计算梯度，并在程序不使用pytorch实现全连接网络梯度计算和迭代过程。
author：JhuangGan

本篇帖子源于ML课程的作业2，查找minist的分类时看到的一篇帖子https://www.kaggle.com/code/scaomath/simple-neural-network-for-mnist-numpy-from-scratch/notebook，当时看到作者手搓全连接梯度迭代，想着一直没手动计算和推导过，打算跟着作者走一遍。
结果发现
（1）一直认为反向传播只是简单的链式法则的应用，忽略了实际计算梯度过程的困难和迭代技巧。
（2）程序实现过程却一直依赖pytorch的黑盒，导致实际中想要变更梯度求导过程以提升性能时，程序一直不知如何下手。
（3）tensor张量的相关知识一直认识不深，比如1*n的tensor对n*m的tensor求梯度，梯度tensor的形状，而其又是如何计算的？当多个高维tensor的梯度出现时，又该如何计算其乘积和链式法则的应用过程是怎么个过程？（这个目前只解决有关该问题的一部分，其余可看文末参考视频）

接下来重新梳理思路，希望以更清晰细致的角度来梳理整个梯度计算过程，最终目的是为弄清楚程序每一步该如何编写，将其实现（因为上面帖子的符号略显混乱，且不够细致，所以下面的符号与上面的帖子不同，将重新设计并说明详细符号代表意义及其维度）。

一、符号说明：
第n层

$Z_n = W^{(n)}O_{n-1}^T + b^{(n)}$
$O_n = f_n(Z_n)$

O_{n-1}为第n-1层的output，维度为$1*S_{n-1}$，Z_n为第n层经过全连接线性层的计算值,维度为$1*S_n$， $W^{(n)}$为第n层的线性算子，维度为S_n*S_{n-1}，$b^{(n)}$为第n层的线性算子，维度为$1*S_n$

O_n为Z_n经过激活函数f_n的输出值,维度为$1*S_n$, n为最终输出层。

第n-1层
$Z_{n-1} = W^{(n-1)}O_{n-2}^T + b^{(n-1)}$
$O_{n-1} = f_{n-1}(Z_{n-1})$

O_{n-2}为第n-2层的output，维度为$1*S_{n-2}$，Z_{n-1}为第n-1层经过全连接线性层的计算值,维度为$1*S_{n-1}$， $W^{(n-1)}$为第n层的线性算子，维度为$S_{n-1}*S_{n-2}$, $b^{(n-1)}$为第n-1层的线性算子，维度为$1*S_{n-1}$

O_{n-1}为Z_{n-1}经过激活函数f_{n-1}的输出值,维度为$1*S_{n-1}$

二、下面从程序角度叙述，神经网络的训练过程：

（1）将样本a，1*k的向量输入，经过第一层的全部k_2个神经元(可看为k_1*1*k_2的tensor)，每个神经元是一个$W_j^{(n)}$维度k_1*1的向量,$b_j^{(n)}$1*1标量,得到一个1*1的标量$Z_{1,1}$, 然后第二个神经元，依此得到第二层k_2个值，再将该$1*k_2$个值组成的$1*k_1*$（k_1*1*k2）$+1*k_2 = 1*k_2$维度的tensor $Z_1, [Z_{1,1}, Z_{1,2}, ..., Z_{1,k_2}]$， 并经过激活函数$f_1$得到$1*k_2$维度的tensor$O_1[O_{1,1}, O_{1,2}, ..., O_{1,k_2}]$。

（2）并继续（1）在下一层的计算，直到得到最终的样本a在Loss function上的loss结果l_a。

（3）将后续样本重复（1）（2）的操作，计算相应的loss l_i。

（4）将所有样本的l_i带入总的Lossfunction计算总Loss。

（5）计算总Loss对于每一层参数W^i, b^i的梯度，可由总$Loss = f(l_1, l_2, ...)$经由链式法则，转换为单样本$loss\ l_a$ 对于每一层参数W^i, b^i的梯度，最终汇总得来，而每一个单样本$loss\ l_a$ 对于每一层参数W^i, b^i的梯度是在每个样本a计算其output o_a后，反向传播计算得来。

（6）每个样本对于每一层参数W^i, b^i的梯度可经由推导证明可由每一层的outputO_i和后续层的当前参数值$W^j, b^j, j < i$所表示。

三、下面推导单样本对每层参数的梯度，并表示为程序可迭代的形式。

l_a

(1.0) 迭代第一步：计算$\frac{\partial l_a}{\partial Z_n}$,$1*S_n$, 然后获得$\frac{\partial l_a}{\partial W^{(n)}}$,$1*S_n*S_{n-1}$和$\frac{\partial l_a}{\partial b^{(n)}}$,$1*S_n$

$\frac{\partial l_a}{\partial Z_n} = \frac{\partial l_a}{\partial O_n}\frac{\partial O_n}{\partial Z_n}, 1*S_n = (1*S_n)*(S_n*S_n) = 1*S_n$

$\frac{\partial l_a}{\partial W^{(n)}} = \frac{\partial l_a}{\partial Z_n}\frac{\partial Z_n}{\partial W^{(n)}}, 1*S_n*S_{n-1} = (1*S_n)*(S_n*S_n*S_{n-1}) = S_n*S_{n-1}$

$\frac{\partial l_a}{\partial b^{(n)}} = \frac{\partial l_a}{\partial Z_n}\frac{\partial Z_n}{\partial b^{(n)}}, 1*S_n = (1*S_n)*(S_n*S_n) = 1*S_n$

(1.1) 各个偏导的具体计算
$\frac{\partial l_a}{\partial O_n}$=$[e_1, e_2, ..., e_{S_n}]$, 因不同loss而不同

例如交叉熵Loss：
$CrossEntropyLoss_a = -\sum_i y_i logO_{n,i}$, 样本a的loss，$y_i$为$a$的$onehot$向量的第$i$个分量（若样本$a$属于第$k$类，则$y_k=1$，其余为0）

，$O_{n,i}$为O_n的第i个分量。
则可以计算得知

$\frac{\partial l_a}{\partial O_{n,i}} = e_i = -\frac{y_i}{O_{n,i}}$

$\rightarrow \frac{\partial l_a}{\partial O_n} = [-\frac{y_1}{O_{n,1}}, -\frac{y_2}{O_{n,2}}, \ldots, -\frac{y_{S_n}}{O_{n,S_n}}]$

将样本a的onehot向量表示为Y_a，那么在python中，可以用numpy.array的广播性质，表示$\frac{\partial l_a}{\partial O_n} = -Y_a/O_n$

```
'''
注意，这里是对一个数据集操作而不是单个样本
y为样本集合的标签值向量，其取值范围为0-Sn，维度为样本量N*1
Y_ont为样本集合的onthot向量，即样本量N*特征数S_n
O_n 为第n层的output，即样本量N*特征数S_n
'''
Y_onehot = (y[:, np.newaxis] == np.arrange(S_n))  #标签向量转为onehot矩阵
PLoverPO = - Y_onehot/O_n  #该矩阵为样本集合的partial li over partial On的梯度矩阵，i行为i样本的梯度向量
```

$\frac{\partial O_n}{\partial Z_n}$，因激活函数不同而不同，

$$\begin{bmatrix} r_{11} & r_{12} & \cdots & r_{1S_n} \\ r_{22} & r_{22} & \cdots & r_{2S_n} \\ \vdots & \vdots & \ddots & \vdots \\ r_{S_n1} & r_{S_n2} & \cdots & r_{S_nS_n} \end{bmatrix}$$

以softmax激活函数为例，$O_{n,i} = \frac{exp(Z_{n,i})}{\sum_k exp(Z_{n,k})}$

$$\frac{\partial O_{n,i}}{\partial Z_{n,j}} = r_{i,j} = \begin{cases} \frac{exp(Z_{n,i})\sum_k exp(Z_{n,k}) - (exp(Z_{n,i}))^2}{(\sum_k exp(Z_{n,k}))^2} = O_{n,i} - O_{n,i}^2，\ i = j \\ -\frac{exp(Z_{n,i})exp(Z_{n,j})}{(\sum_k exp(Z_{n,k}))^2} = -O_{n,i}O_{n,j}，\ i \neq j \end{cases}$$

```
def POoverPZ_softmax(On):
    '''
    输入：On
    输出：partial On over partial Zn 的matrix
    ''' # 如果是要分开使用的话，可以写该函数。但如果是一些固定搭配，比如CEloss与softmax的经典组合，将上面两个整合后计算，更省计算资源。此处略
```

(1.2) 将上面两部分整合后再写代码更容易，比如softmax和softmax组合后，

$$\left[-\frac{y_1}{O_{n,1}}, -\frac{y_2}{O_{n,2}}, \ldots, -\frac{y_{S_n}}{O_{n,S_n}}\right] \times \begin{bmatrix} O_{n,1} - O_{n,1}^2 & -O_{n,1}O_{n,2} & \cdots & -O_{n,1}O_{n,S_n} \\ -O_{n,2}O_{n,1} & O_{n,2} - O_{n,2}^2 & \cdots & -O_{n,2}O_{n,S_n} \\ \vdots & \vdots & \ddots & \vdots \\ -O_{n,S_n}O_{n,1} & -O_{n,S_n}O_{n,2} & \cdots & -O_{S_nS_n} - O_{S_nS_n}^2 \end{bmatrix}$$

$$= \left[O_{n,1}(\textstyle\sum_k y_k) - 1, O_{n,2}(\sum_k y_k) - 1, \ldots, O_{n,S_n}(\sum_k y_k) - 1\right]$$

$$= \left[O_{n,1} - 1, O_{n,2} - 1, \ldots, O_{n,S_n} - 1\right]$$

即,使用CEloss与softmax时，$\frac{\partial l_a}{\partial Z_n} = O_n - 1_{1*S_n}$

```
PLoverPZ = On - 1
```

(1.3) 计算$\frac{\partial l_a}{\partial W^{(n)}}$与$\frac{\partial l_a}{\partial b^{(n)}}$

$\frac{\partial l_a}{\partial W^{(n)}} = \frac{\partial l_a}{\partial Z_n}\frac{\partial Z_n}{\partial W^{(n)}}, 1 * S_n * S_{n-1} = (1 * S_n) * (S_n * S_n * S_{n-1}) = S_n * S_{n-1}$

$\frac{\partial Z_n}{\partial W^{(n)}}$,维度为$S_n * S_n * S_{n-1}$,如下所示，由$S_n$个$S_n * S_{n-1}$矩阵组成，即$S_n * S_n * S_{n-1}$的tensor

$Z_n = [Z_{(n,i)}], i = 1, 2, \ldots, S_n, W^{(n)} = [W_{i,j}^{(n)}], i = 1, 2, \ldots, S_n, j = 1, 2, \ldots, S_{n-1}$

$$\begin{bmatrix} O_{n-1,1} & O_{n-1,2} & \cdots & O_{n-1,S_{n-1}} \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 0 & \cdots & 0 \\ O_{n-1,1} & O_{n-1,2} & \cdots & O_{n-1,S_{n-1}} \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix}$$

$$\cdots$$

$$\begin{bmatrix} 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \\ O_{n-1,1} & O_{n-1,2} & \cdots & O_{n-1,S_{n-1}} \end{bmatrix}$$

$$\frac{\partial l_a}{\partial W^{(n)}} = \frac{\partial l_a}{\partial Z_n}\frac{\partial Z_n}{\partial W^{(n)}} = [e_1, e_2, ..., e_{S_n}]\frac{\partial Z_n}{\partial W^{(n)}}$$

$$= \begin{bmatrix} e_1 O_{n-1,1} & e_1 O_{n-1,2} & \cdots & e_1 O_{n-1,S_{n-1}} \\ e_2 O_{n-1,1} & e_2 O_{n-1,2} & \cdots & e_2 O_{n-1,S_{n-1}} \\ \vdots & \vdots & \ddots & \vdots \\ e_{S_n} O_{n-1,1} & e_{S_n} O_{n-1,2} & \cdots & e_{S_n} O_{n-1,S_{n-1}} \end{bmatrix}$$

$$= [e_i O_{n-1,j}], i = 1, 2, ..., S_n, j = 1, 2, ..., S_{n-1}$$

$$= \begin{bmatrix} e_1 \\ e_2 \\ \cdots \\ e_{S_n} \end{bmatrix} \times \left[O_{n-1,1}, O_{n-1,1}, ..., O_{n-1,S_{n-1}}\right]$$

$$\rightarrow \frac{\partial l_a}{\partial W^{(n)}} = [e_i]^T * O_{n-1}$$

```
'''
PloverPZ 为partial la over partial Zn, 对于样本集合的，维度为样本量N*S_{n-1}
O_(n-1)为第n-1层output，维度为样本量N*S_{n-1}
'''
# 如果总LOSS = \frac{1}{n}\sum_{i}l_i, i=1,2,...,n
的话，下面可以直接如此计算，不然的话，要对每个样本分别计算其梯度矩阵，分别存储，最后经过f(l1,l2,...)的计算进行汇总为总LOSS，
PloverPW = np.matmul(PloverPZ.T * O_(n-1))/N
```

$$\frac{\partial l_a}{\partial b^{(n)}} = \frac{\partial l_a}{\partial Z_n}\frac{\partial Z_n}{\partial b^{(n)}}, 1 * S_n = (1 * S_n) * (S_n * S_n) = 1 * S_n$$

$\frac{\partial Z_n}{\partial b^{(n)}}$容易证明其为一个$S_n * S_n$的单位对角阵，所以$\frac{\partial l_a}{\partial b^{(n)}} = \frac{\partial l_a}{\partial Z_n}$

(2.0) 迭代第二步，计算$\frac{\partial l_a}{\partial Z_{n-1}}$,$1 * S_{n-1}$, 然后获得$\frac{\partial l_a}{\partial W^{(n-1)}}$,$1 * S_{n-1} * S_{n-2}$和$\frac{\partial l_a}{\partial b^{(n-1)}}$,$1 * S_{n-1}$

$$Z_n = W^{(n)} O_{n-1}^T + b^{(n)}$$
$$O_n = f_n(Z_n)$$
$$Z_{n-1} = W^{(n-1)} O_{n-2}^T + b^{(n-1)}$$
$$O_{n-1} = f_{n-1}(Z_{n-1})$$

$\frac{\partial l_a}{\partial Z_{n-1}} = \frac{\partial l_a}{\partial Z_n}\frac{\partial Z_n}{\partial O_{n-1}}\frac{\partial O_{n-1}}{\partial Z_{n-1}} = \frac{\partial l_a}{\partial Z_n} W^{(n)} \frac{\partial O_{n-1}}{\partial Z_{n-1}}$, $1 * S_{n-1} = (1 * S_n) * (S_n * S_{n-1}) * (S_{n-1} * S_{n-1}) = 1 * S_{n-1}$，易知，$\frac{\partial Z_n}{\partial O_{n-1}} = W^{(n)}$

$\frac{\partial O_{n-1}}{\partial Z_{n-1}}$与第n-1层激活函数选择有关。
如果选择softmax激活函数，其梯度矩阵为1.1中所示形式。

如果选择relu激活函数，其梯度矩阵函数如下：

```
def relu(O):
    '''
    输入：O为某一层output, S_n-k * S_n-k-1
    输出：relu计算的梯度（\frac{\partial O_{n-1}}{\partial Z_{n-1}}）
    '''
    O[O<0] = 0
    O[O>0] = 1
    # 或者
    O = 1*(O>0)
    return O
```

下面以第n-1层激活函数为relu为例，继续

$\rightarrow \frac{\partial l_a}{\partial W^{(n-1)}} = \frac{\partial l_a}{\partial Z_{n-1}} \frac{\partial Z_{n-1}}{\partial W^{(n-1)}} = (\frac{\partial l_a}{\partial Z_{n-1}})^T * O_{n-2}$

由1.3可知，$\frac{\partial l_a}{\partial Z_{n-1}} \frac{\partial Z_{n-1}}{\partial W^{(n-1)}} = (\frac{\partial l_a}{\partial Z_{n-1}})^T * O_{n-2}$

且$\frac{\partial l_a}{\partial b^{(n-1)}} = \frac{\partial l_a}{\partial Z_{n-1}}$

总上所述，将n和n-1层的参数迭代公式整理如下：

$$\frac{\partial l_a}{\partial Z_n} = \frac{\partial l_a}{\partial O_n} \frac{\partial O_n}{\partial Z_n} \rightrightarrows \frac{\partial l_a}{\partial W^{(n)}} = (\frac{\partial l_a}{\partial Z_n})^T * O_{n-1}, \frac{\partial l_a}{\partial b^{(n)}} = \frac{\partial l_a}{\partial Z_n}$$
$$\frac{\partial l_a}{\partial Z_{n-1}} = \frac{\partial l_a}{\partial Z_n} W^{(n)} \frac{\partial O_{n-1}}{\partial Z_{n-1}} \rightrightarrows \frac{\partial l_a}{\partial W^{(n-1)}} = (\frac{\partial l_a}{\partial Z_{n-1}})^T * O_{n-2}, \frac{\partial l_a}{\partial b^{(n-1)}} = \frac{\partial l_a}{\partial Z_{n-1}}$$

可以清晰的递归得到除第n层外其他层的参数更新迭代公式为：

$$\frac{\partial l_a}{\partial Z_{n-k}} = \frac{\partial l_a}{\partial Z_{n-k+1}} W^{(n-k+1)} \frac{\partial O_{n-k}}{\partial Z_{n-k}} \rightrightarrows \frac{\partial l_a}{\partial W^{(n-k)}} = (\frac{\partial l_a}{\partial Z_{n-k}})^T * O_{n-k-1}, \frac{\partial l_a}{\partial b^{(n-k)}} = \frac{\partial l_a}{\partial Z_{n-k}}$$

k = 1, 2, ...

从迭代公式上，很容易可知，当确定第n-k层的激活函数的梯度向量$\frac{\partial O_{n-k}}{\partial Z_{n-k}}$, 由上一层计算得到的$\frac{\partial l_a}{\partial Z_{n-k+1}}$与此时的线性变化层$W^{(n-k+1)}$，即可计算得到n-k层的$\frac{\partial O_{n-k}}{\partial Z_{n-k}}$，也即该层对于常数项$b^(n-k)$的梯度向量；再与n-k-1层的output，可计算得到对于该层线性变化层的梯度矩阵$\frac{\partial l_a}{\partial W^{(n-k)}}$，从而不断迭代下去。

最终结局：这个是推导完了，但我的作业还没开始做啊啊啊，不务正业一天天的。
还是挺有意思的，并不是像之前想象中的那么无序，而是能够通过迭代实现梯度的更新。后续如果可以将各种常见loss的梯度计算为函数，激活函数也像relu一样计算梯度函数，那么就可以实现大部分的梯度计算，而不通过pytorch。这样想想pytorch里封装的东西大抵应该也是这样计算的，有规律的，感觉也可以写OOP的程序来实现，不过这种有生之年系列，再说吧，不如等个几年找个空闲再看看pytorch源码来的直接。

参考资料：
https://www.kaggle.com/code/scaomath/simple-neural-network-for-mnist-numpy-from-scratch/notebook

tensor视频链接：https://www.youtube.com/@eigenchris/playlists 博主的tensor for beginners & tensor calculus两个视频很不错，但太长了没看完，希望之后有时间能看完。