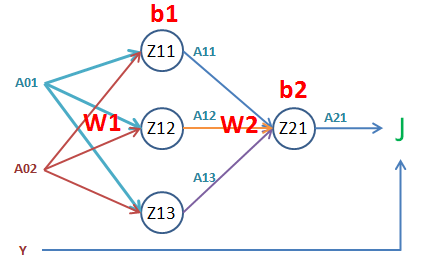
# 再聊BP网络的实现

再次写一个BP网络的程序，主要说明反向传播的原理及实现。

## 网络的结构



以第一层来说，若输入一组样本数量为N，则各矩阵的形状如下：

对于流动的样本有关的数据，每行是一个样本，每列是样本的一个特征，如这里样本的特征数量为2。

## 前向传播

前向传播运算过程：

最后计算Cost，用J表示，以交叉熵为例，计算如下：

## 反向传播

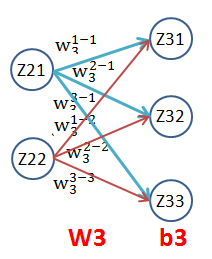
反向传播运算过程：

对最后一层，最后的节点使用Sigmod函数作为激活函数，然后计算交叉熵损失，所以本层的导数如下：

以上各元素皆为矩阵，\*表示对应元素相乘，为N行1列，因为这里只有一个神经元输出。Y为输入的样本标签，N行1列。A为本层的输出，为N行M列，其中M为神经元的数量。以上与样本数据有关的矩阵都是N行，与输入样本对应。dW和db不与样本相关，只与网络结构有关，大小是固定的，dW是NL-1行NL列，db是1行M列，M为本层神经元的数量。

对于其它层，有以下递推公式：

例如：



对Z31节点分析，因为

故有

**L层的梯度等于L+1层梯度乘以本层神经元输出的权重再乘以本层激活函数的导数**。

同理也有：

故有：

进而有：

用矩阵表示：

权重的计算：

## 代码实现

实验使用的数据集在这里下载：

https://pan.baidu.com/s/1fjvN3ujV3hXSfY7lkMhO1g

|  |
| --- |
| **import** h5py  **import** matplotlib**.**pyplot **as** plt  **import** numpy **as** np  np**.**random**.**seed**(**0**)** # 用于指定随机数生成时所用算法开始的整数值，如果使用相同的seed值，则每次生成的随机数相同  **def** load\_data**():**  # 1.创建文件对象  train\_dataset **=** h5py**.**File**(**'./data/train\_catvnoncat.h5'**,** "r"**)** # 调用h5py库里面的File方法，创建一个文件对象  test\_dataset **=** h5py**.**File**(**'./data/test\_catvnoncat.h5'**,** "r"**)**  # 3.根据键名获取键值，即获取我们的图片数据  train\_set\_x\_orig **=** np**.**array**(**train\_dataset**[**"train\_set\_x"**][:])** # 获取训练集中的特征数据x  train\_set\_y\_orig **=** np**.**array**(**train\_dataset**[**"train\_set\_y"**][:])** # 获取训练集中的标签数据y  test\_set\_x\_orig **=** np**.**array**(**test\_dataset**[**"test\_set\_x"**][:])** # 获取测试集中的特征数据x  test\_set\_y\_orig **=** np**.**array**(**test\_dataset**[**"test\_set\_y"**][:])** # 获取测试集中的标签数据y  classes **=** np**.**array**(**test\_dataset**[**"list\_classes"**][:])** # 获取标签名称列表cat 和 non-cat  # 4.reshape训练集和测试集标签的维数  train\_set\_y\_orig **=** train\_set\_y\_orig**.**reshape**((**1**,** train\_set\_y\_orig**.**shape**[**0**]))**  test\_set\_y\_orig **=** test\_set\_y\_orig**.**reshape**((**1**,** test\_set\_y\_orig**.**shape**[**0**]))**  **return** train\_set\_x\_orig**,** train\_set\_y\_orig**.**reshape**(-**1**,** 1**),** test\_set\_x\_orig**,** test\_set\_y\_orig**.**reshape**(-**1**,** 1**),** classes  **def** sigmoid**(**Z**):**  **return** 1 **/** **(**1 **+** np**.**exp**(-**Z**))**  **def** init\_net**(**net**):**  **print(**"init net"**)**  parameters **=** **{}**  layer\_num **=** len**(**net**)**  **for** l **in** range**(**1**,** layer\_num**):**  parameters**[**"W" **+** str**(**l**)]** **=** np**.**random**.**randn**(**net**[**l **-** 1**],** net**[**l**])** **\*** 0.1  parameters**[**"b" **+** str**(**l**)]** **=** np**.**zeros**([**1**,** net**[**l**]])**  **for** l **in** range**(**1**,** layer\_num**):**  **print(**"W" **+** str**(**l**)** **+** " shape = "**,** parameters**[**"W" **+** str**(**l**)].**shape**)**  **print(**"b" **+** str**(**l**)** **+** " shape = "**,** parameters**[**"b" **+** str**(**l**)].**shape**)**  **return** parameters  **def** preprocess**(**input**):**  **return** **(**input **-** 128.0**)/**255.0  **def** forward**(**A**,** parameters**):**  cache **=** **{}**  cache**[**"A0"**]** **=** A  l **=** 1  layer\_num **=** len**(**parameters**)** **//** 2  **for** l **in** range**(**1**,** layer\_num**):**  Z **=** np**.**dot**(**A**,** parameters**[**"W" **+** str**(**l**)])**  A **=** sigmoid**(**Z**)**  cache**[**"Z" **+** str**(**l**)]** **=** Z  cache**[**"A" **+** str**(**l**)]** **=** A  Z **=** np**.**dot**(**A**,** parameters**[**"W" **+** str**(**l **+** 1**)])**  A **=** sigmoid**(**Z**)**  cache**[**"Z" **+** str**(**l **+** 1**)]** **=** Z  cache**[**"A" **+** str**(**l **+** 1**)]** **=** A  **return** A**,** cache  **def** backward**(**AL**,** Y**,** parameters**,** cache**):**  m **=** Y**.**shape**[**0**]**  gradient **=** **{}**  layer\_num **=** len**(**parameters**)** **//** 2  dZL **=** **-** **(**np**.**divide**(**Y**,** AL**)** **-** np**.**divide**(**1 **-** Y**,** 1 **-** AL**))** **\*** AL **\*** **(**1 **-** AL**)**  gradient**[**"dW" **+** str**(**layer\_num**)]** **=** np**.**dot**(**cache**[**"A" **+** str**(**layer\_num **-** 1**)].**T**,** dZL**)/**m  gradient**[**"db" **+** str**(**layer\_num**)]** **=** np**.**sum**(**dZL**,** axis**=**0**,** keepdims**=True)** **/** m  **for** l **in** reversed**(**range**(**1**,** layer\_num**)):**  dZL **=** np**.**dot**(**dZL**,** parmeter**[**"W" **+** str**(**l **+** 1**)].**T**)** **\*** cache**[**"A" **+** str**(**l**)]** **\*** **(**1 **-** cache**[**"A" **+** str**(**l**)])**  gradient**[**"dW" **+** str**(**l**)]** **=** np**.**dot**(**cache**[**"A" **+** str**(**l **-** 1**)].**T**,** dZL**)** **/** m  gradient**[**"db" **+** str**(**l**)]** **=** np**.**sum**(**dZL**,** axis**=**0**,** keepdims**=True)** **/** m  **return** gradient  **def** update**(**parameter**,** gradient**,** learning\_rate **=** 0.01**):**  layer\_num **=** len**(**parameter**)** **//** 2  **for** l **in** range**(**layer\_num**):**  parameter**[**"W" **+** str**(**l **+** 1**)]** **=** parameter**[**"W" **+** str**(**l **+** 1**)]** **-** learning\_rate **\*** gradient**[**"dW" **+** str**(**l **+** 1**)]**  parameter**[**"b" **+** str**(**l **+** 1**)]** **=** parameter**[**"b" **+** str**(**l **+** 1**)]** **-** learning\_rate **\*** gradient**[**"db" **+** str**(**l **+** 1**)]**  **return** parmeter  **def** compute\_cost**(**AL**,** Y**):**  m **=** Y**.**shape**[**0**]**  cost **=** **-**np**.**sum**(**Y **\*** np**.**log**(**AL**)** **+** **(**1 **-** Y**)** **\*** np**.**log**(**1 **-** AL**))** **/** m  cost **=** np**.**squeeze**(**cost**)**  **return** cost  train\_x\_orig**,** train\_y**,** test\_x\_orig**,** test\_y**,** classes **=** load\_data**()**  net **=** **[**12288**,** 20**,** 7**,** 5**,** 1**]**  parmeter **=** init\_net**(**net**)**  batch\_size **=** 209  cost\_list **=** **[]**  **for** epoch **in** range**(**400**):**  batch\_num **=** len**(**train\_y**)** **//** batch\_size  cost **=** 0  **for** i **in** range**(**batch\_num**):**  x **=** train\_x\_orig**[**i **\*** batch\_size**:(**i **+** 1**)** **\*** batch\_size**]**  y **=** train\_y**[**i **\*** batch\_size**:(**i **+** 1**)** **\*** batch\_size**]**  x **=** x**.**reshape**([-**1**,** 12288**])**  x **=** preprocess**(**x**)**  al**,** cache **=** forward**(**x**,** parmeter**)**  cost **=** compute\_cost**(**al**,** y**)**  gradient **=** backward**(**al**,** y**,** parmeter**,** cache**)**  update**(**parmeter**,** gradient**,** learning\_rate**=**0.2**)**  cost\_list**.**append**(**cost**)**  plt**.**plot**(**cost\_list**)**  plt**.**show**()** |

