目录

[layer模块 1](#_Toc508960362)

[data类 1](#_Toc508960363)

[fusion\_layer类 1](#_Toc508960364)

[fully\_connected\_layer类 2](#_Toc508960365)

[activation\_layer类 3](#_Toc508960366)

[loss\_layer类 3](#_Toc508960367)

[function\_for\_layer模块 4](#_Toc508960368)

[激活函数： 4](#_Toc508960369)

[损失函数： 4](#_Toc508960370)

[初始化函数： 5](#_Toc508960371)

[update\_method模块 5](#_Toc508960372)

[学习率变化机制函数： 5](#_Toc508960373)

[权值更新机制： 5](#_Toc508960374)

# layer模块

## data类

**属性：**

data\_sample：用于保存样本集，为的二维数组，*S*表示样本数，N表示一个样本的分量个数。

data\_sample: 用于保存样本标签，为的二维数组，*S*表示样本数，N表示样本中总的类别数，例如样本集中类别总数为10，其中一个样本属于每4类，则它的标签为：。

output\_sample:用于保存下次训练使用的样本，为的二维数组，*b*等于batch\_size的大小，N表示一个样本的分量个数。

output\_label: 用于保存下次训练使用的样本标签。

**方法：**

\_\_init\_\_(self): 构造函数。

get\_data(self, sample, label): 获得样本集与对应的标签。

shuffle(self): 对样本集进行洗牌，即打乱它们的顺序。

pull\_data(self): 用于从训练集中推出下一次训练使用的样本与标签。

## fusion\_layer类

**属性：**

num\_dimension: 一个样本的分量个数*N*。

inputs1: 保存特征1的样本集；的二维数组，*b*表示batch\_size的大小，*N*表示样本分量。

inputs2: 保存特征2的样本集。

inputs3: 保存特征3的样本集。

outputs: 三种特征加权融合后的特征。

weights: 保存加权系数；的一维数组。为什么是2个权系数呢，因为设定了这三个加权系数和为1。

previous\_direction: 保存两个加权系数上一次的下降方向，用于带加动量向的权值更新；的一维数组。

grad\_weights: 保存加权系数的梯度；的二维数组，*b*表示bach\_size的大小。

grad\_outputs\_now: 保存本层输出的梯度；的二维数组，*N*表示本层输出的神经元个数，即样本分量了。

**方法：**

\_\_init\_\_(self, num\_dimension)：构造函数。

initialize\_weights(self, weight1, weight2): 初始化加权系数。

get\_inputs\_for\_forward(self, input1, input2, input3): 获取正向传播的输入。

forward(self): 对三种输入进行加权求和，得到输出。



get\_inputs\_for\_backward(self, grad\_outputs): 获取反向传播的输入。

backward(self): 求两个加权系数的梯度。



update(self): 计算多个样本的平均梯度，更新权值。

## fully\_connected\_layer类

**属性：**

num\_neuron\_inputs: 输入层的神经元个数*M*。

num\_neuron\_outputs: 本层的神经元个数*N*。

inputs: 本层输入***I***；的二维数组。 *b*表示batch\_size的大小。

outputs: 本层的输出***O***；的二维数组。

weights: 权值***W***；的二维数组。

bias: 偏置***b***；的一维数组。

weights\_previous\_direction: 上一次的下降方向；的二维数组。

bias\_previous\_direction: 上一次的下降方向；的一维数组。

grad\_weights: 权值的梯度；的三维数组。

grad\_bias: 偏置的梯度；的二维数组。

grad\_inputs: 输入的梯度；的二维数组。

grad\_outputs: 输出的梯度；的二维数组。

**方法：**

\_\_init\_\_(self, num\_neuron\_inputs, num\_neuron\_outputs): 构造函数。

initialize\_weights(self): 初始化权值。

get\_inputs\_for\_forward(self, inputs): 获取正向传播输入。

forward(self): 计算输出值。

get\_inputs\_for\_backward(self, grad\_outputs): 获取反向传播输入。

backward(self): 求权值与偏置的梯度。







update(self): 计算样本的平均梯度，更新权值与偏置。

## activation\_layer类

**属性：**

activation\_function: 使用的激活函数。

der\_activation\_function: 使用的激活函数的导数。

inputs: 输入。

outputs:输出。

grad\_inputs: 输入的梯度。

grad\_outputs: 输出的梯度。

**方法：**

\_\_init\_\_(self, activation\_function\_name): 构造函数。

get\_inputs\_for\_forward(self, inputs): 获取正向传播输入。

forward(self): 利用激活函数求输出的值。



get\_inputs\_for\_backward(self, grad\_outputs): 获取反向传播输入。

backward(self): 利用激活函数的导数求输入的导数。



## loss\_layer类

**属性：**

inputs: 训练样本的输入。

loss: 训练误差。

accuracy: 正确率。

label: 训练样本的标签。

grad\_inputs: 输入的梯度。

loss\_function: 使用的损失函数。

der\_loss\_function: 损失函数的导数。

**方法：**

\_\_init\_\_(self, loss\_function\_name): 构造函数。

get\_labe\_for\_loss(self, label): 获取训练样本的标签。

get\_inputs\_for\_loss(self, inputs): 获取输入。

compute\_loss\_and\_accuracy(self): 计算训练误差与正确率。

compute\_gradient(self): 计算输入的梯度。

# function\_for\_layer模块

## 激活函数：

sigmoid（*x*）: sigmoid函数。



der\_sigmoid(x): sigmoid函数的导数。



tanh(x): tanh函数。



der\_tanh(x): tanh函数的导数。



relu(x): relu函数。



der\_relu(x): 函数的数。



## 损失函数：

softmaxwithloss(inputs, label):



der\_softmaxwithloss(inputs, label):



上式中，*a*表示样本的标签为第*a*类，*z*表示每*z*个输出。

## 初始化函数：

xavier(num\_neuron\_inputs, num\_neuron\_outputs): 按xavier方法初始化。

权值服从均匀：

# update\_method模块

## 学习率变化机制函数：

inv(gamma = 0.0005, power = 0.75 ):



式中，lr0 表示初始学习率， *n* 表示迭代次数， gamma 与*p* 为参数。

fixed():



## 权值更新机制：

batch\_gradient\_descent(weights, grad\_weights, previous\_direction): 基于批量的随机梯度下降法。