目录

layer 模块

data 类

属性：

data_sample：用于保存样本集，为 $S \times N$ 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 $S \times N$ 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：$[0,0,0,1,0,0,0,0,0,0]$。

output_sample:用于保存下次训练使用的样本，为 $b \times N$ 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 $\boldsymbol{I}_1$； $b \times N$ 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 $\boldsymbol{I}_2$。

inputs3: 保存特征 3 的样本集 $\boldsymbol{I}_3$。

outputs: 三种特征加权融合后的特征 $\boldsymbol{O}$ 。
weights: 保存加权系数；1×2 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；1×2 的一维数组。
grad_weights: 保存加权系数的梯度；$b \times 2$ 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；$b \times N$ 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。

$$\boldsymbol{O} = w_1 \cdot \boldsymbol{I}_1 + w_2 \cdot \boldsymbol{I}_2 + (1 - w_1 - w_2) \cdot \boldsymbol{I}_3$$

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。

$$\frac{\partial J}{\partial k_1} = \sum_{i=1}^{N} \left[\frac{\partial J}{\partial O_i^{(\text{本层的输出})}} \cdot \left(x_i^1 - x_i^3 \right) \right] + \lambda \cdot k_1$$

$$\frac{\partial J}{\partial k_2} = \sum_{i=1}^{N} \left[\frac{\partial J}{\partial O_i^{(\text{本层的输出})}} \cdot \left(x_i^2 - x_i^3 \right) \right] + \lambda \cdot k_2$$

其中x_i^1表示I_1 的第i个分量，λ表示权值衰减系数。

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 $\boldsymbol{I}$； $b \times M$ 的二维数组。 b 表示 batch_size 的大小。
outputs: 本层的输出 $\boldsymbol{O}$； $b \times N$ 的二维数组。
weights: 权值 $\boldsymbol{W}$； $M \times N$ 的二维数组。
bias: 偏置 $\boldsymbol{b}$； $1 \times N$ 的一维数组。
weights_previous_direction: 上一次的下降方向； $M \times N$ 的二维数组。
bias_previous_direction: 上一次的下降方向； $1 \times N$ 的一维数组。
grad_weights: 权值的梯度； $b \times M \times N$ 的三维数组。
grad_bias: 偏置的梯度； $b \times N$ 的二维数组。
grad_inputs: 输入的梯度； $b \times M$ 的二维数组。
grad_outputs: 输出的梯度； $b \times N$ 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。 $\boldsymbol{O} = \boldsymbol{I} \bullet \boldsymbol{W} + \begin{bmatrix} \boldsymbol{b} \\ \vdots \\ \boldsymbol{b} \end{bmatrix}$，　　其中 $\bullet$ 表示矩阵乘法

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。

$$\frac{\partial J}{\partial w_{ij}^{(L)}} = \frac{\partial J}{\partial O_j^{(L)}} \cdot \frac{\partial O_j^{(L)}}{\partial w_{ij}^{(L)}} + \lambda \cdot w_{ij}^{(L)} = I_i^{(L)} \cdot \frac{\partial J}{\partial O_j^{(L)}} + \lambda \cdot w_{ij}^{(L)}$$

$$\frac{\partial J}{\partial b_j^{(L)}} = \frac{\partial J}{\partial O_j^{(L)}} \cdot \frac{\partial O_j^{(L)}}{\partial b_j^{(L)}} = \frac{\partial J}{\partial O_j^{(L)}}$$

$$\frac{\partial J}{\partial I_i^{(L)}} = \sum_{j=1}^{N_L} \left[\frac{\partial J}{\partial O_j^{(L)}} \cdot \frac{\partial O_j^{(L)}}{\partial I_j^{(L)}} \right] = \sum_{j=1}^{N_L} \left[\frac{\partial J}{\partial O_j^{(L)}} \cdot w_{ij}^{(L)} \right]$$

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。

$$\boldsymbol{O} = f(\boldsymbol{I})$$

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。

$$\frac{\partial J}{\partial \boldsymbol{I}} = \frac{\partial J}{\partial \boldsymbol{O}} \cdot \frac{\partial \boldsymbol{O}}{\partial \boldsymbol{I}} = \frac{\partial J}{\partial \boldsymbol{O}} \cdot f'(\boldsymbol{I})$$

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。

$$f(x)=\frac{1}{1+e^{-x}}$$

der_sigmoid(x): sigmoid 函数的导数。

$$f'(x)=f(x)\cdot[1-f(x)]$$

tanh(x): tanh 函数。

$$f(x)=\frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$$

der_tanh(x): tanh 函数的导数。

$$f(x)=1-[f(x)]^{2}$$

relu(x): relu 函数。

$$f(x)=\begin{cases}x, & x\geq 0\\ 0, & x<0\end{cases}$$

der_relu(x): 函数的数。

$$f'(x)=\begin{cases}1, & x\geq 0\\ 0, & x<0\end{cases}$$

损失函数：

softmaxwithloss(inputs, label):

$$J=-\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{j=1}^{k}\mathrm{I}\left\{\mathrm{T}^{(i)}=j\right\}\ln\frac{\mathrm{e}^{y_j^{(i)}}}{\sum_{l=1}^{k}\mathrm{e}^{y_l^{(i)}}}\right]$$

der_softmaxwithloss(inputs, label):

$$\frac{\partial J_1}{\partial O_z^{(L)}}=-\frac{\partial\log(\frac{\mathrm{e}^{O_a^{(L)}}}{\sum_{l=1}^{4}\mathrm{e}^{O_l^{(L)}}})}{\partial O_z^{(L)}}=\frac{\mathrm{e}^{O_z^{(L)}}}{\sum_{l=1}^{4}\mathrm{e}^{O_l^{(L)}}}-\delta_{az},\quad \text{其中}\delta_{az}=\begin{cases}1 & z=a\\ 0 & z\neq a\end{cases}$$

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：$W\sim U\left[-\frac{\sqrt{6}}{\sqrt{n_j+n_{j+1}+1}},\frac{\sqrt{6}}{\sqrt{n_j+n_{j+1}+1}}\right]$

update_method 模块

学习率变化机制函数：

inv(gamma = 0.0005, power = 0.75):

$$lr=lr_0\times\left(1+\gamma\times n\right)^{-p}$$

式中，lr_0 表示初始学习率， n 表示迭代次数， gamma 与 p 为参数。

fixed():

$$lr=lr_0$$

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。