智能系统原理与开发 Lab1

贺劲洁　18307130370

代码基本架构

-- myNetwork.py

数据读取、结果图像绘制、不同任务（拟合、分类及测试）的主函数

-- myTraining.py

神经网络功能函数

NetWork类：

可自由组合激活函数、损失函数，能对网络结构进行灵活调整、可自动停止训练并记录最佳参数

`__init__()`

神经网络初始化，设置神经网络结构参数和功能（拟合或分类）

【参数说明】

- **layers**:

  每层节点数数组

- **func_type**:
  - fit : 二次代价
  - classfy： 交叉熵 + softmax
  - classify_bi: 二次代价 + sigmoid
  - cross_sigmoid: 交叉熵 + sigmoid
- **init_weight**，**init_bias**

  调整权重、偏置的初始化范围

`stochastic_gradient_descent( )`

参数：raining_set, epochs, batch_size,rate, lmbda，beta，validation_set, validate_freq, stop_cnt

传入训练相关的参数，采用随机梯度下降法进行训练，可通过参数选择梯度更新策略，可灵活设置验证频率。

梯度更新函数：

regulation_update() 使用正则项的梯度更新

momentum_update() 使用动量优化的梯度更新

反向传播算法函数

通过反向传播算法计算每次的参数更新量

1. **quadraticCost_BP_delta ()**

   使用二次代价函数（拟合问题最后一层不过 sigmoid 函数

2. **crossEntropyCost_BP_delta ()**

   使用交叉熵代价函数 （分类问题最后一层用 softmax ）

网络参数实验比较及思考

激活函数

sigmoid VS softmax

每一层可以设置不同的激活函数

1. sigmoid

输出范围 [0,1]

2. softmax

适用于分类问题，因能简化运算常与 cross entropy 搭配使用

【理解】

1. 作用：将输入向量归一化映射到一个类别概率分布
2. hardmax：将真正最大的以100%的概率被选出来

   softmax：所有的都有机会被作为最大值选出来，大的几率更大，小的几率更小

3. 为什么分类问题最后一层要用softmax?

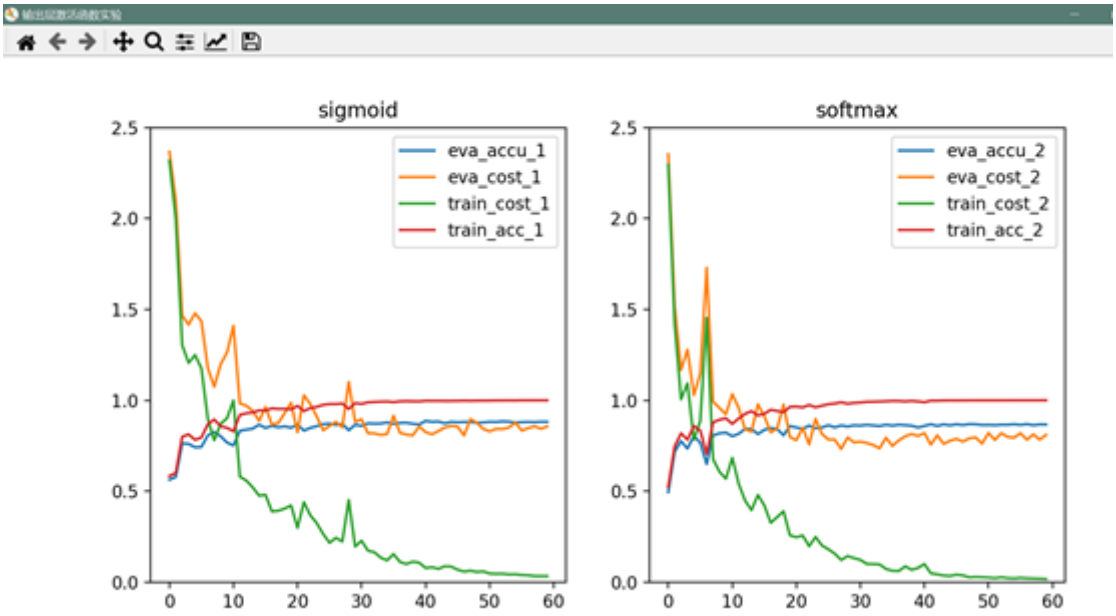
sigmoid两侧导数逐渐趋近于0，当该层输入值较大时，sigmoid的导数会使梯度减小，使网络参数难以得到有效更新，出现梯度消失的情况

sigmoid只能使每个节点的输出为[0,1]， 而softmax使最后一层输出值相互关联，总和为1，增加一类的概率，其他类的概率就会降低

4. 实验对比：softmax + cross 和 sigmoid + cross

输出层是否采用 `softmax`

使用 softmax 的收敛效果更好、loss更低



代价函数

平方代价函数 VS 交叉熵代价函数

二次代价函数—— 一般用于拟合任务

交叉熵代价函数—— 一般用于分类任务

1. 思考：为什么拟合问题一般用平方代价函数，而分类问题用交叉熵?

交叉熵：$Loss = \sum I\{output = expect\} * ln(1 - output)$

从公式可以看出，交叉损失函数对正确的结果更加看重，而平方损失函数还和错误的结果有关。

若在分类问题中使用交叉熵误差函数，相当于调整除了让正确分类尽量变大，还让错误分类更加平均，但这个调整对于分类问题实际上是没必要的。但对于拟合问题来说，这一调整是有必要的。

2. 分类问题中，交叉熵代价函数的优势

对于采用sigmoid作为激活函数的层来说，

$\delta_w = a_{l-1} \cdot \delta_{l+1} \cdot w_{l+1} \cdot \sigma'(z)$

由于$\sigma'(z)$ 的存在，当神经元输出接近0时，δ_w 和 δ_b 的值会很小，导致

交叉熵函数导数的一项可以将sigmoid导数抵消，从而：

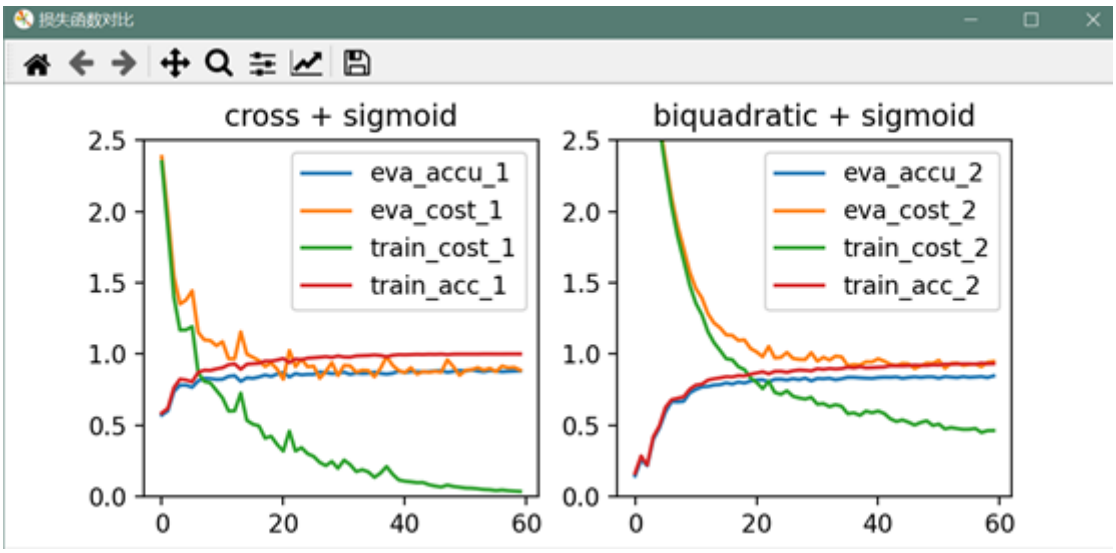
$\frac{\partial L}{\partial w} = z(a-y)$

$\frac{\partial L}{\partial b} = a - y$

可见参数的调整量由 output-expect 控制，误差越大，调整越大，学习越快

3. 实验：cross + sigmoid VS quadratic + sigmoid

（正确率：cross：0.8875，quadratic：0.8458



可观察到，使用二次代价函数的收敛速度更慢、收敛效果更差、训练集的loss更高

思考：关于 loss 和 accuracy 的关系

如果正确分类的概率降低，但仍然是概率最高的，会出现损失增加但准确度不变的结果

如果数据集的标签不平均，比如某种分类更多，可能出现正确率上升， 但loss增大

从实际用途上来看，对测试集的评估应该以准确度为准

Batch-Size

1. **梯度下降的分类：**

- full batch

  每次遍历整个数据集后进行梯度更新

  优点：每次更新的梯度更准确，收敛过程稳定

  缺点：可能收敛到初始点附近的局部最优解，难以跳出

- 随机梯度下降

  对每个训练样本的每个输入神经元都要计算梯度

  缺点：收敛波动较大

  优点：可以利用波动较大的特点增加跳出局部极值的概率

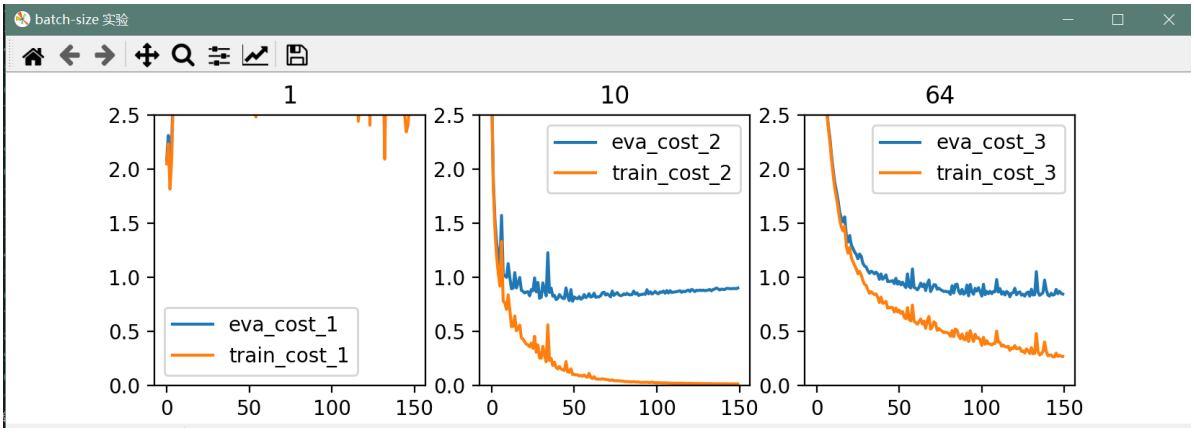
- mini-batch

  前两个方法的折中，每次抽取样本中的一份计算梯度然后更新，同时具有防止过拟合的作用

2. batch-size 的选择

随着 batch_size 增大，处理相同数据量的速度加快，但收敛速度更慢，因此需要相应地增大学习率

3. 实验对比



在相同学习率下，batch-size越大，收敛速度越慢；

而由于0.05的学习率对于batch-size为1的过大，因此无法收敛

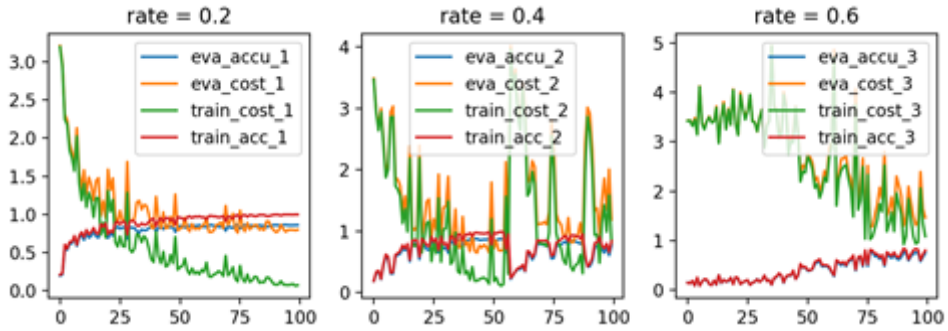
（分析见下方学习率部分）

learning-rate（学习率）

学习率过小，收敛太慢，过大，收敛波动大

batch-size小时，过大的学习率会使模型更难收敛，且易受到噪声的影响

1. 实验对比：学习率0.2 vs 0.4 vs 0.6



（其他参数：batch-size = 64， λ = 0

可观察到，随着学习率的升高，收敛波动加大甚至无法收敛

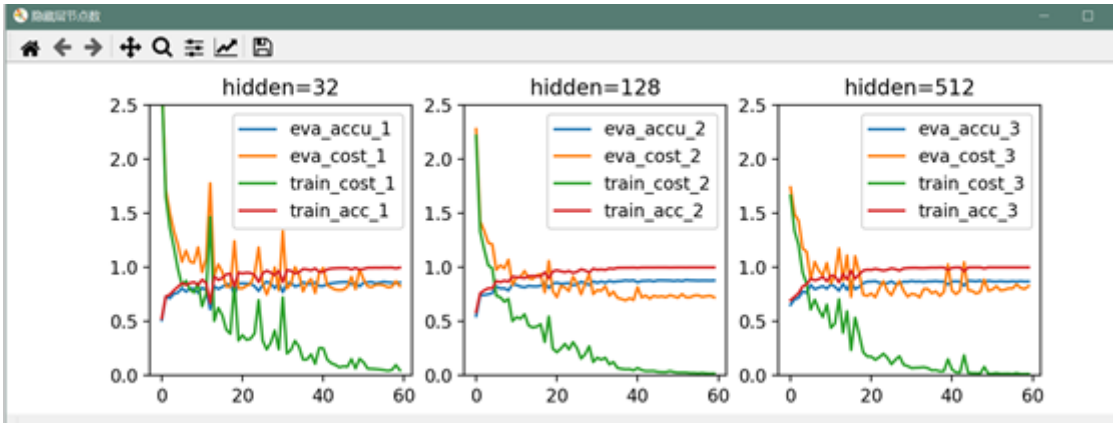
2. 思考：batch-size 和 学习率的关系

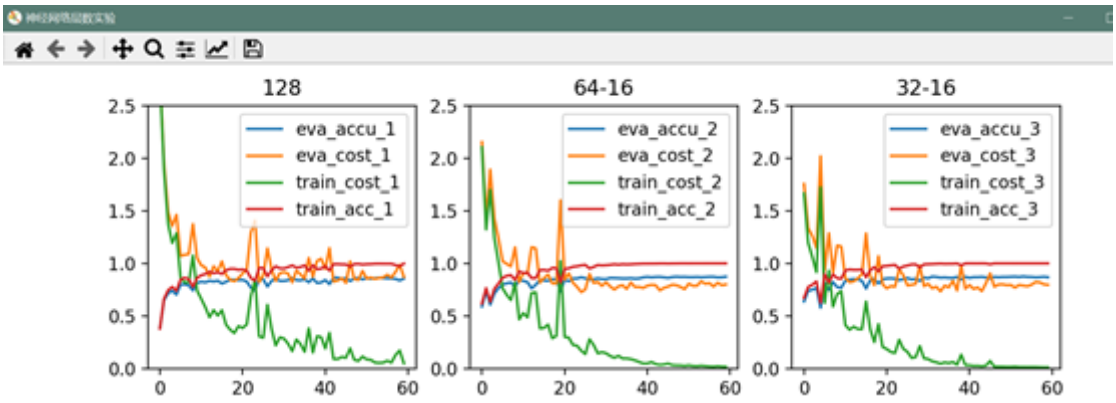
学习率要随着 batch-size 的增加而增加，当 batch-size 较小时，如果学习率过大会导致收敛波动大、噪声影响大的问题

当batch-size较大时，梯度更加准确，可通过较大的学习率提高收敛速度，过小的学习率会导致收敛过慢

神经网络结构

实验：





神经元个数

神经元更多可以表达更复杂的函数，但是过多可能造成训练数据的过拟合。同时，神经元个数太高不仅对准确率的提升没有作用，还会使训练速度、程序性能大大降低。

学习的epoch数

提前停止：准确率不再提升时终止（防止过度拟合）

本网络采用10个epoch内，训练集的准确率提高不超过0.001则自动停止训练。

λ （正则项系数）

1. 理解

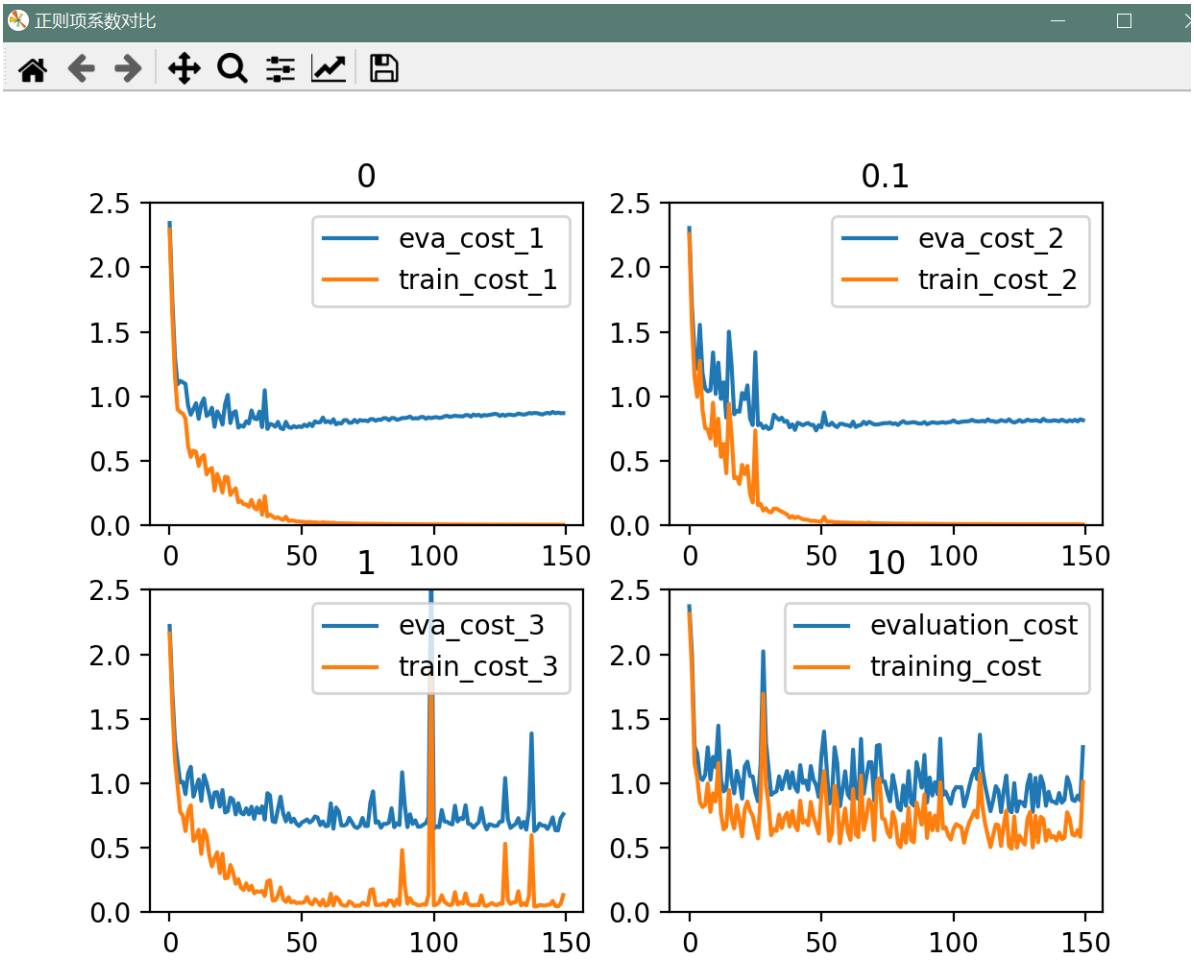
作用：拟合过程中让权值参数尽可能小，在一定程度上减少过拟合情况。

分析：参数值小的模型对不同的数据集的适应性更强，因为如果参数很大，只要数据有一点偏移就可能对结果产生很大的影响。加入正则项后，w除了减小更新量，还要乘以一个小于1的因子（$1-\alpha\frac{\lambda}{n}$）。

注：

1. λ 越大，模型泛化性越强，但太大可能造成欠拟合
2. 正则项系数可以随着梯度的改变而改变，过拟合的时候可以适当加大系数，非过拟合的时候可不使用或不做调整
3. 神经网络的正则化项是所有隐藏层和输出层上所有神经元的正则化之和

2. 实验对比：λ = 0, 0.1, 1, 10



观察发现:

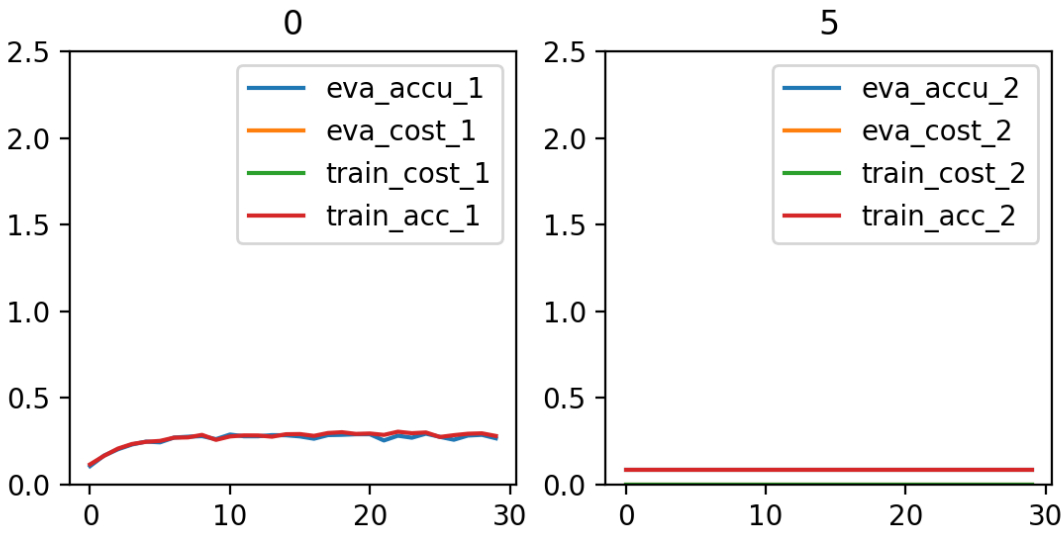
1. 不使用正则项优化时，当训练次数较高时，验证集的 loss 逐渐提高，出现了过拟合情况；
2. 随着正则项系数 λ 增大，收敛速度也有所减慢，但当λ系数过大时（λ = 10) 时，将严重影响训练收敛效果

权重初始化范围

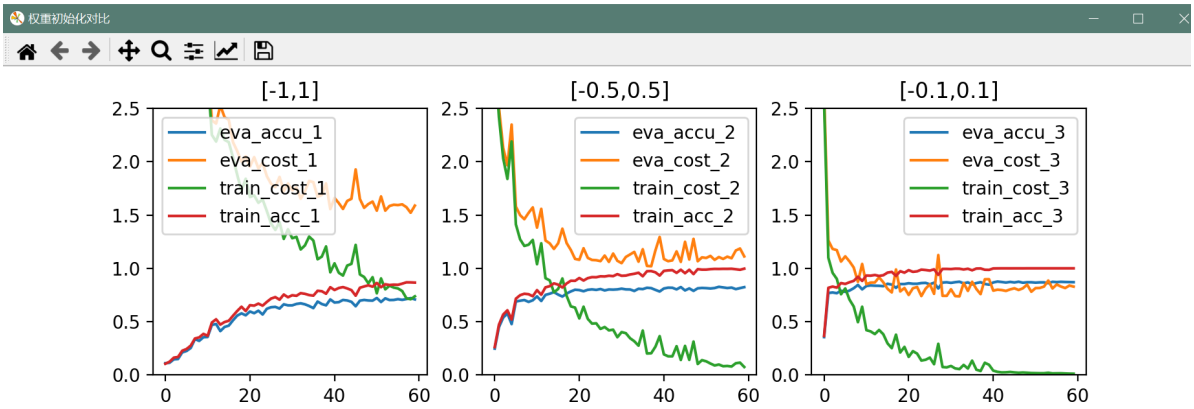
1. 分析

如果权重初始化为0，用梯度下降算法失效，因为每个神经元输出结果相同，会计算出同样的梯度、得到完全相同的参数更新，不收敛

如果初始化的范围过大（比如一开始选用正态分布函数）或过小，会导致sigmoid值逼近 1 或 0 ，梯度很小，出现梯度消失的问题，不收敛



2. 实验对比：正态分布*1，0.5，0.1



对反向传播算法的理解

每一层节点都依赖于上一层的节点和权重

$$
\begin{gathered}
\text{第}l\text{层偏置}：b_l \\
\text{第}l-1\text{层和第}l\text{层之间的权重}：\ w_l \\
\text{输入}：z_l = w_l \cdot a_{l-1} + b_l \\
\text{输出}：a_l = activation(z_l)
\end{gathered}
$$

定义 *L* 为 误差函数

$$
\text{令}：\text{第}l\text{层的误差}：\delta_l = \frac{\partial L}{\partial z_l} = \frac{\partial L}{\partial a_l} \cdot \frac{\partial a_l}{\partial z_l}, \text{则}：
$$

$$
\delta_{w_l} = \frac{\partial L}{\partial w_l} = \frac{\partial L}{\partial a_l} \cdot \frac{\partial a_l}{\partial z_l} \cdot \frac{\partial z_l}{\partial w_l}
$$

$$
\frac{\partial z_j}{\partial w_j} = \frac{\partial (w_l \cdot a_{l-1} + b_l)}{\partial w_j} = a_{l-1}
$$

$$
\frac{\partial z_j}{\partial b_j} = 1
$$

$$
\begin{gathered}
\text{第}l\text{层 } weight \text{ 误差}：\delta_{w_l} = \delta_l \cdot a_{l-1} \\
\text{第}l\text{层 } bias \text{ 误差}：\delta_{b_l} = \delta_l
\end{gathered}
$$

又因为

$$
\delta_l = \frac{\partial L}{\partial z_l} = \frac{\partial L}{\partial z_{l+1}} \cdot \frac{\partial z_{l+1}}{\partial z_l}
$$

$$
= \delta_{l+1} \cdot \frac{\partial w_{l+1} a_l + b_{l+1}}{\partial a_l} \cdot \frac{\delta a_l}{\delta z_l}
$$

$$
= \delta_{l+1} \cdot w_{l+1} \cdot \frac{\delta a_l}{\delta z_l}
$$

因此，从输出层向前，每一层的误差梯度都可以通过下一层的误差推到得出，即反向传播

优化策略：

1. **动量优化**

现象：使用mini-batch时，loss有时候会出现激增

分析：由于mini-batch是分批次计算梯度进行调整，有可能朝错误方向突进

作用：减小训练调整的波动减小噪声的干扰

原理：类似增加一个物理中的“惯性”，在每一次调整的时候把之前的更新方向也考虑进来，这样能减缓大幅度的更新

使用指数加权平均的方式，引入一个“速度”的概念，当许多连续的梯度指向相同的方向时，更新最大

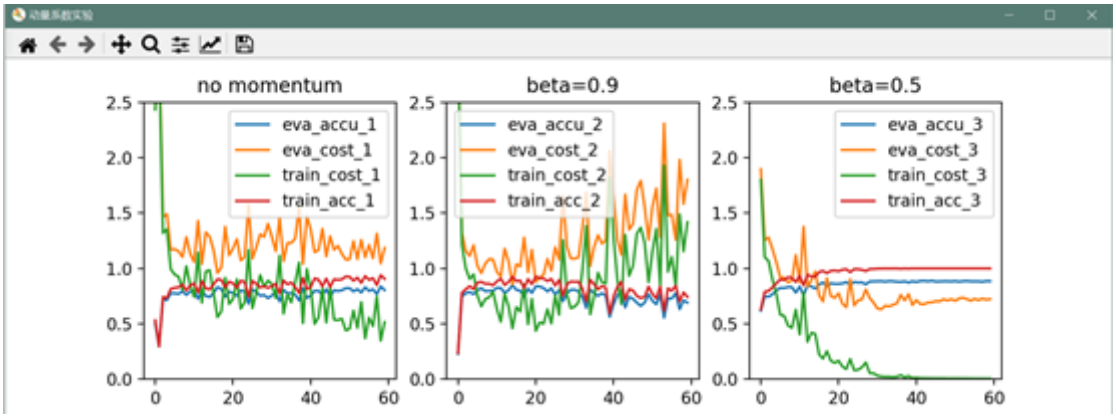
$v_i = \beta \cdot v_{i-1} - rate * \delta w$

$new\_w = w + v$

如果之前的 v 和这次的负梯度方向相同，下降的幅度就会加大，从而加速收敛

否则下降的幅度减小，减少大幅度的错误更新

实验：适当调大学习率，对比 $\beta = 0.5$、0.9 和不使用动量



可见当 $\beta = 0.5$ 时，收敛波动程度减小、正常收敛，但β 过大也会有副作用

2. 其他优化

1. 可选的 正则项
2. 记录正确率，10个epoch正确率提升不超过0.001时提前停止训练，防止过拟合
3. 记录测试集正确率最高时的参数
4. 学习率衰减：经过一定epoch数后减小学习率，减小后期在最优值附近的震荡，使更好地收敛到最优值（根据对曲线的观察，选取收敛曲线波动开始明显增大的节点开始适当调低学习率）