一、原理

反向传播算法基于多元函数的链式法则。而链式法则可以使用雅各比矩阵来表达。

雅各比矩阵的定义如下：

假设 F 是一个从 n 维空间映射到 m 维空间的函数，即该函数由 m 个实函数组成。则这些函数的偏导数可以组成一个 m 行 n 列的矩阵：

$$\begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_m}{\partial x_1} & \cdots & \frac{\partial y_m}{\partial x_n} \end{bmatrix}.$$

可以用符号表达为 JF(x1…xn)。

那么显然，如果有函数(y1…ym)=F(x1…,xn)和函数(z1…zk)=G(y1…ym).则：

Z 对 x 的雅各比矩阵等于 z 对 y 的雅各比矩阵乘 y 对 x 的雅各比矩阵。即 z 对 x 的偏导的链式法则。

二、实现

由以上知识可以知道，只需在正向传播中保留当前的雅各比矩阵即可。反向传播的时候将各层雅各比矩阵反向相乘即可，这一过程正好与正向传播相反。对于 softmax 和 relu 本身不需要参数，所以直接把梯度向后传即可。而对于线性层则需要不仅保存输出对输入的梯度用来反向传播，还需要保存输出对参数的梯度，用来自身梯度下降。因为使用 batch 来做训练，所以应该对 batch 中的每一个样本保留梯度，而一个样本梯度是二维矩阵，如此应要保存三维张量。使用 Eigen 作为矩阵运算库。由于该库只有二维矩阵，不支持三维张量，所以只能使用对二维矩阵做循环来实现计算三维张量，也就是说不能使用广播机制来计算。除此以外，由于对 Eigen 了解较少，所以某些地方并没有实现较好的优化。

实现的层将 Module 作为基类，所有模型继承自 Module，基类提供一系列虚函数，由子类 override。Loss 类作为损失函数的基类，交叉熵损失函数继承 Loss。

DataLoader 作为数据加载类。

Trian 类包括训练的业务逻辑和测试的业务逻辑。

代码：[TeRiRi2333/mnist-eigen (github.com)](https://github.com/TeRiRi2333/mnist-eigen)

三、实验

1. 模型结构

Layer1	全连接层(784×128)
Layer2	Relu

Layer3	全连接层(128×64)
Layer4	Relu
Layer5	全连接层(64×10)
Layer6	Softmax

2. 训练参数

训练样本大小	60000
测试集大小	10000
学习率	0.01
Batch_size	128
epoches	2
学习率下降策略	每 epoch 乘 0.9
损失函数	交叉熵损失

3. 训练结果

训练集 Loss	0.011
训练集 Acc	99.2%
测试集 Loss	0.094
测试集 Acc	97.1%
训练时间	42min