梯度下降法与反向传播

七月算法 龙老师

2016年5月21日

主要内容

- 梯度下降法
  1. 损失函数可视化
  2. 最优化
  3. 梯度下降
- 反向传播
  1. 梯度与偏导
  2. 链式法则
  3. 直观理解
  4. Sigmoid例子

预备知识

- 两个重要函数
  - 得分函数
  - 损失函数
- 核心目标
  - 找到最合适的参数w.
  - 使得损失函数取值最小化。
  - 也就是最优化的过程

$f = w * x$

$L =$

损失函数可视化

- 损失函数往往定义在非常高维的空间
  - 比如CIFAR-10的例子里一个线性分类器的权重矩阵W是10 x 3073维的，总共有30730个参数

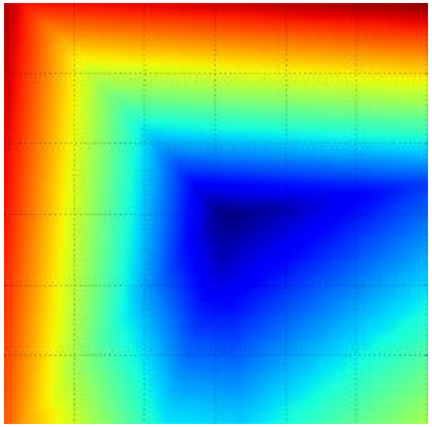
- 曲线救国
  - 我们可以把高维投射到一个向量/方向(1维)或者一个面(2维)上，从而能直观地『观察』到一些变化

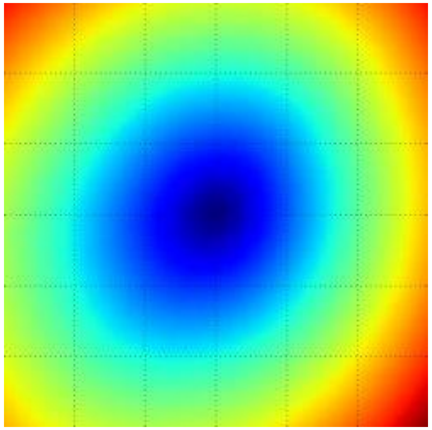
损失函数可视化

- 举个栗子
  - 我们找到一个方向W1(维度要和W一样),
  - 然后我们给不同的a值，计算L(W+aW1)

损失函数可视化

- 第二个栗子
  - 我们给两个方向W1和W2，那么我们可以确定一个平面，我们再取不同值的a和b，计算L(W+aW1+bW2)的值。





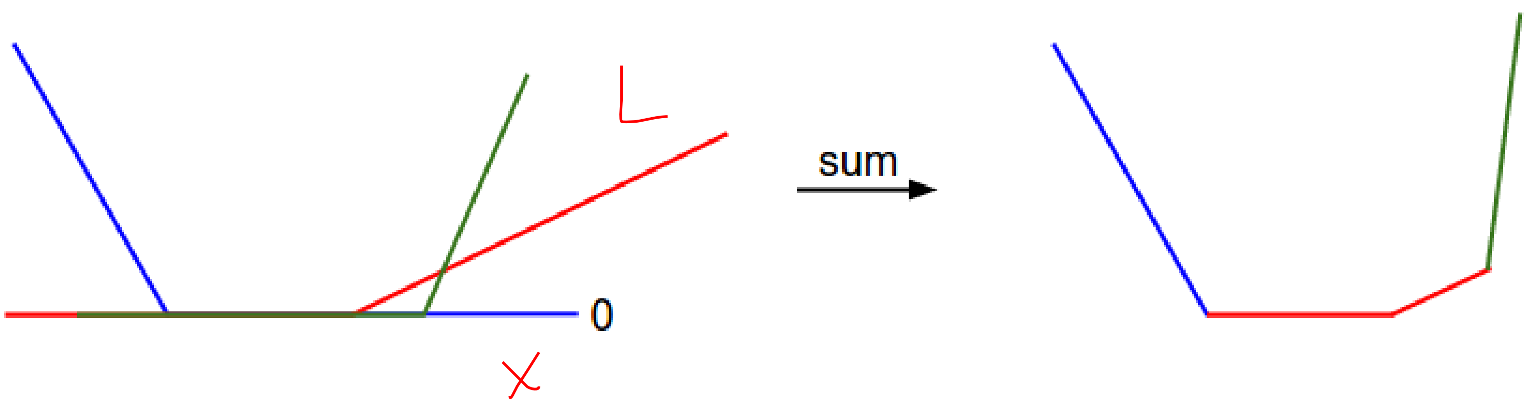
损失函数可视化

- 假定训练集里面有3个样本，都是1维的，同时总共有3个类别。其SVM损失：

$$L_0 = \max\ (0, w_1^T x_0 - w_0^T x_0 + 1) + \max(0, w_2^T x_0 - w_0^T x_0 + 1)$$
$$L_1 = \max\ (0, w_0^T x_1 - w_1^T x_1 + 1) + \max(0, w_2^T x_1 - w_1^T x_1 + 1)$$
$$L_2 = \max\ (0, w_0^T x_2 - w_2^T x_2 + 1) + \max(0, w_1^T x_2 - w_2^T x_2 + 1)$$
$$L = (L_0 + L_1 + L_2)/3$$

损失函数可视化

- 假定训练集里面有3个样本，都是1维的，同时总共有3个类别。其SVM损失：



凸优化

- SVM损失函数是一个凸函数。
- 凸函数的正系数加和仍然是凸函数。
- 但扩充到神经网络之后，损失函数将变成一个非凸函数

最优化

最优化：

- 策略1：随机搜寻(不太实用)
  - 最直接粗暴的方法就是，我们尽量多地去试参数，然后从里面选那个让损失函数最小的，作为最后的W。

```
# 假设 X_train 是训练集 (例如. 3073 x 50,000)
# 假设 Y_train 是类别结果 (例如. 1D array of 50,000)

bestloss = float("inf") # 初始化一个最大的float值
for num in xrange(1000):
  W = np.random.randn(10, 3073) * 0.0001 # 随机生成一组参数
  loss = L(X_train, Y_train, W) # 计算损失函数
  if loss < bestloss: # 比对已搜寻中最好的结果
    bestloss = loss
    bestW = W
  print 'in attempt %d the loss was %f, best %f' % (num, loss, bestloss)
```

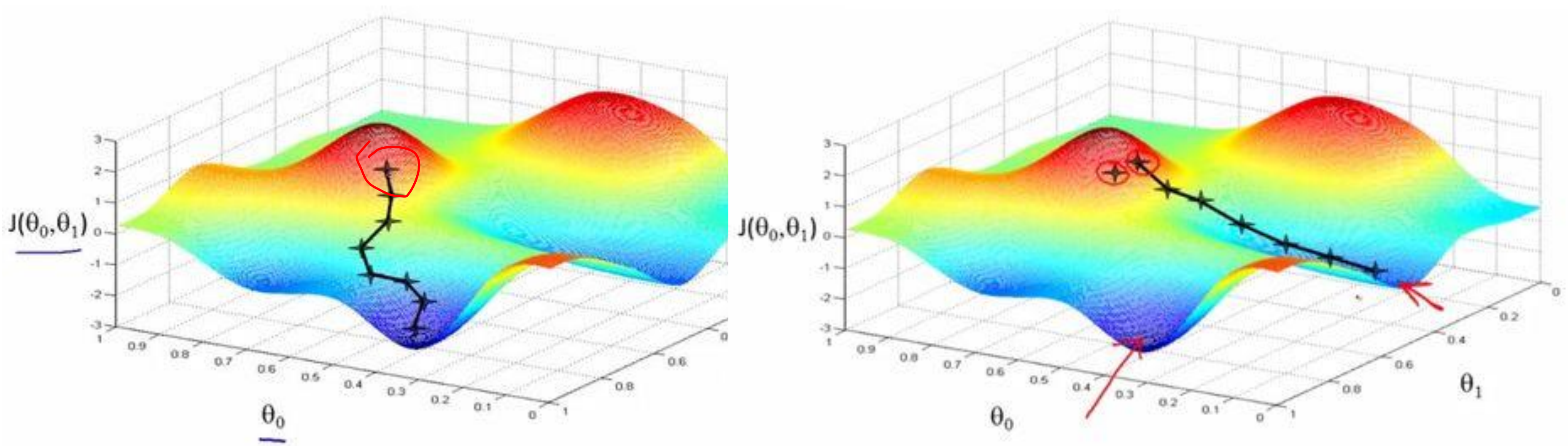
最优化：

- 策略2：随机局部搜索
  - 在现有的参数W基础上，搜寻一下周边临近的参数，有没有比现在参数更好的W，然后我们用新的W替换现在的W，不断迭代。

```
W = np.random.randn(10, 3073) * 0.001 # 初始化权重矩阵W
bestloss = float("inf")
for i in xrange(1000):
  step_size = 0.0001
  Wtry = W + np.random.randn(10, 3073) * step_size
  loss = L(Xtr_cols, Ytr, Wtry)
  if loss < bestloss:
    W = Wtry
    bestloss = loss
  print 'iter %d loss is %f' % (i, bestloss)
```

最优化：

- 策略3：顺着梯度下滑
  - 找到最陡的方向，迈一小步，然后再找当前位置最陡的下山方向，再迈一小步…



计算梯度

- 有两种计算梯度的方法：
  - 慢一些但是简单一些的数值梯度/numerical gradient
  - 速度快但是更容易出错的解析梯度/analytic gradient

$$\frac{f(x+h)-f(x)}{h}$$

$$\frac{\partial f}{\partial x} = w$$

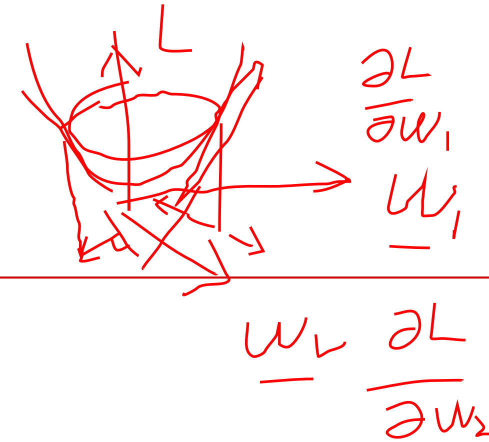
计算梯度

- 数值梯度：
- 对每个维度，都在原始值上加上一个很小的h，然后计算这个维度/方向上的偏导，最后组在一起得到梯度grad。

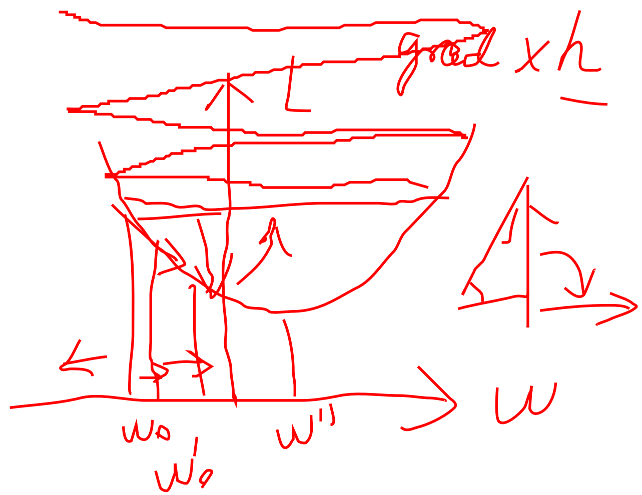
```
def eval_numerical_gradient(f, x):
  """
  一个最基本的计算x点上f的梯度的算法
  - f 为参数为x的函数
  - x 是一个numpy的vector
  """

  fx = f(x) # 计算原始点上函数值
  grad = np.zeros(x.shape)
  h = 0.00001

  # 对x的每个维度都计算一遍
  it = np.nditer(x, flags=['multi_index'], op_flags=['readwrite'])
  while not it.finished:

    # 计算x+h处的函数值
    ix = it.multi_index
    old_value = x[ix]
    x[ix] = old_value + h # 加h
    fxh = f(x) # 计算f(x + h)
    x[ix] = old_value # 存储之前的函数值

    # 计算偏导数
    grad[ix] = (fxh - fx) / h # 斜率
    it.iternext() # 开始下一个维度上的偏导计算

  return grad
```

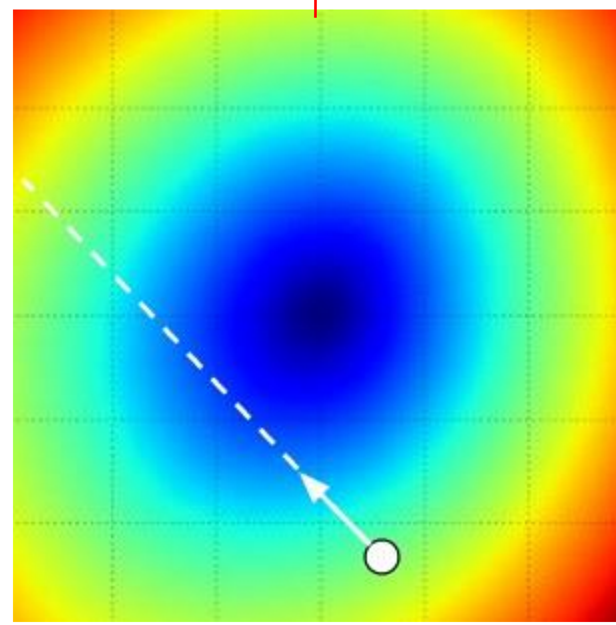
$$\lim_{h \to 0} \frac{f(x+h)-f(x)}{h}$$

计算梯度



- 关于迭代的步长：
  - 步子迈得太小，时间耗不起。
  - 步子迈得太大，容易……





计算梯度

- 关于效率问题：
  - 这个计算方法的复杂度，基本是和我们的参数个数成线性关系的。
  - 在CIFAR-10例子中，我们总共有30730个参数
  - 这个问题在神经网络中更为严重，很可能两层神经元之间就有百万级别的参数权重。
  - 人也要等结果等到哭瞎...

计算梯度

- 解析法计算梯度：
  - 速度非常快
  - 但是容易出错
    - 反倒之前的数值法就显出优势。
  - 我们可以先计算解析梯度和数值梯度，然后比对结果和校正，然后就可以大胆地进行解析法计算了
    - 这个过程叫做梯度检查/检测

计算梯度

- 一个样本点的SVM损失函数：

$$L_i = \sum_{j \neq y_i} \left[\max(0, w_j^T x_i - w_{y_i}^T x_i + \Delta) \right]$$

- 求偏导：

$$\nabla_{w_{y_i}} L_i = -\left(\sum_{j \neq y_i} 1(w_j^T x_i - w_{y_i}^T x_i + \Delta > 0) \right) x_i$$

梯度下降

- 这个简单的循环就是很多神经网络库的核心：

```python
while True:
  weights_grad = evaluate_gradient(loss_fun, data, weights)
  weights += - step_size * weights_grad # 梯度下降更新参数
```

梯度下降

- Mini-batch：
  - 对整个训练数据集的样本都算一遍损失函数，以完成参数迭代是一件非常耗时的事情，一个我们通常会用到的替代方法是，采样出一个子集在其上计算梯度。

```
while True:
  data_batch = sample_training_data(data, 256) # 抽样256个样本作为一个batch
  weights_grad = evaluate_gradient(loss_fun, data_batch, weights)
  weights += - step_size * weights_grad # 参数更新
```

第二部分：反向传播

- 偏导与梯度的关系
- 链式法则：若函数 $u=\varphi(t), v=\psi(t)$ 在点 t 可导, $z=f(u,v)$

$$\frac{\mathrm{d}z}{\mathrm{d}t}=\frac{\partial z}{\partial u}\cdot\frac{\mathrm{d}u}{\mathrm{d}t}+\frac{\partial z}{\partial v}\cdot\frac{\mathrm{d}v}{\mathrm{d}t}$$

$$\frac{\partial f}{\partial w}=\frac{\partial z}{\partial w}\times\frac{\partial f}{\partial z}$$

链式法则

- 比如函数f(x,y,z)=(x+y)*z：

$f = q \times z$

q

```
x = -2; y = 5; z = -4

# 前向计算
q = x + y # q becomes 3
f = q * z # f becomes -12

# 类反向传播:
# 先算到了 f = q * z
dfdz = q # df/dz = q
dfdq = z # df/dq = z
# 再算到了 q = x + y
dfdx = 1.0 * dfdq # dq/dx = 1 恩，链式法则
dfdy = 1.0 * dfdq # dq/dy = 1
```

$\frac{\partial f}{\partial q}$ $\frac{\partial f}{\partial z}$

$\frac{\partial f}{\partial x}$ $\frac{\partial q}{\partial x}$ $\frac{\partial q}{\partial y}$

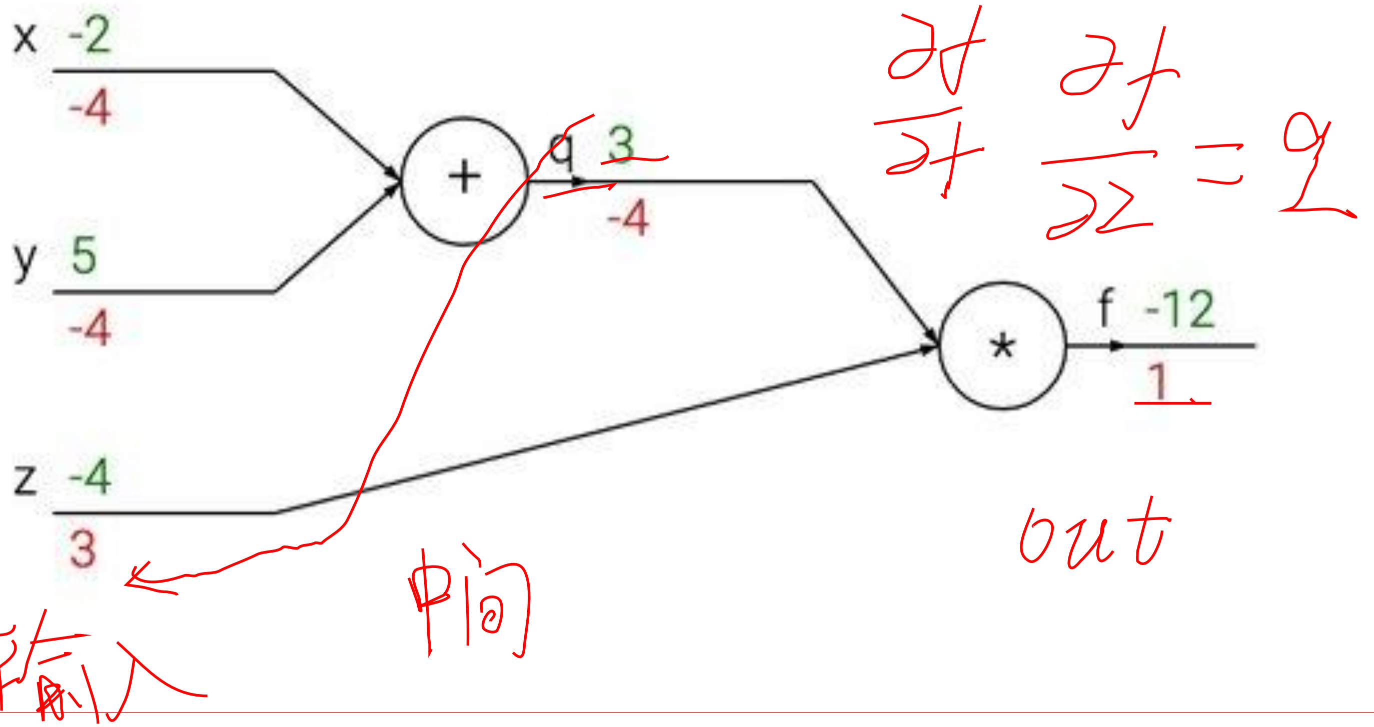
$\frac{\partial f}{\partial y}$

链式法则

- 比如函数f(x,y,z)=(x+y)*z：

$\frac{\partial f}{\partial x}$ $\frac{\partial f}{\partial y}$ $\frac{\partial f}{\partial z}$

$\frac{\partial f}{\partial f}$ $\frac{\partial f}{\partial z} = q$



Sigmoid例子

- 函数：

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$

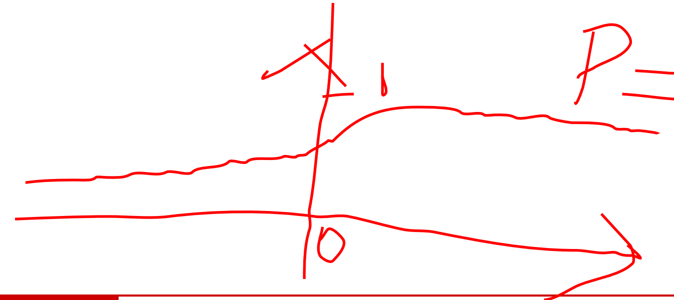
$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

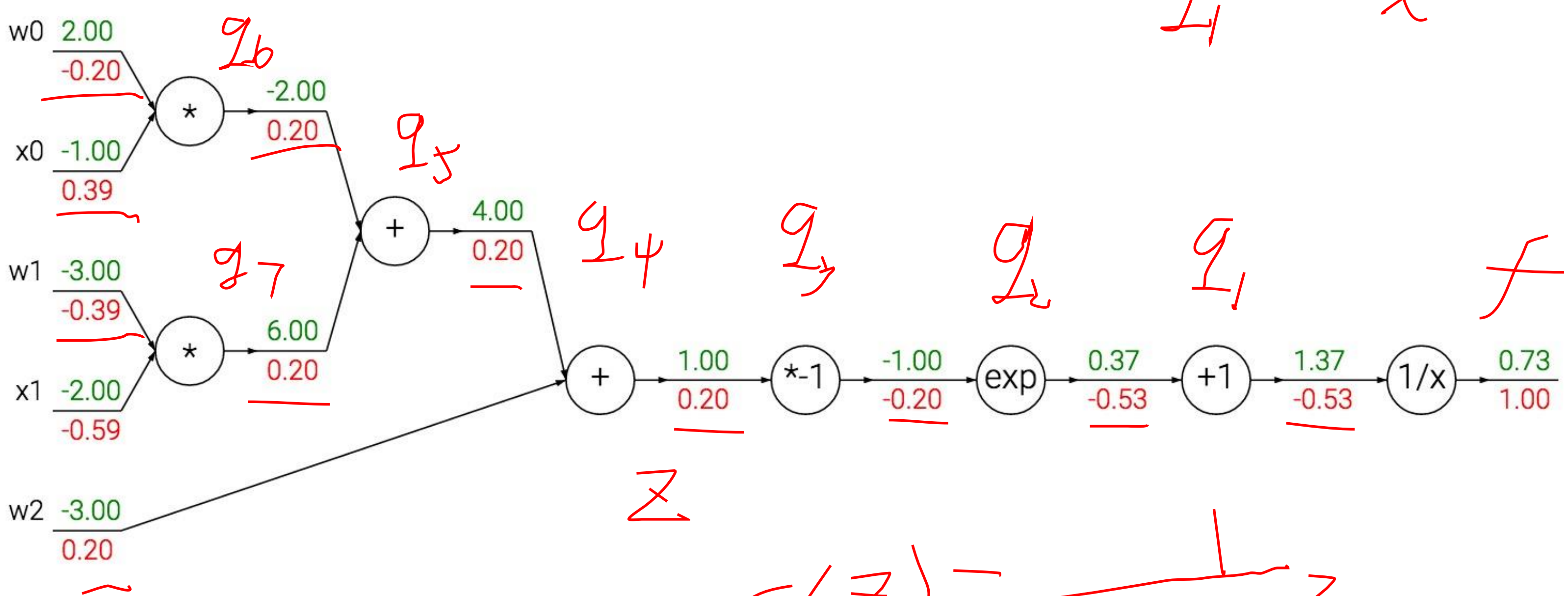
$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

Sigmoid例子



$$\frac{\partial f}{\partial q_1} = -\frac{1}{x^2}$$



$$\sigma(z) = \frac{1}{1+e^{-z}}$$

Sigmoid例子

- Sigmoid函数的导数可以用自己很简单的重新表示出来(非常重要):

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

$$\frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1+e^{-x})^2} = \left(\frac{1+e^{-x}-1}{1+e^{-x}}\right)\left(\frac{1}{1+e^{-x}}\right) = (1-\sigma(x))\sigma(x)$$

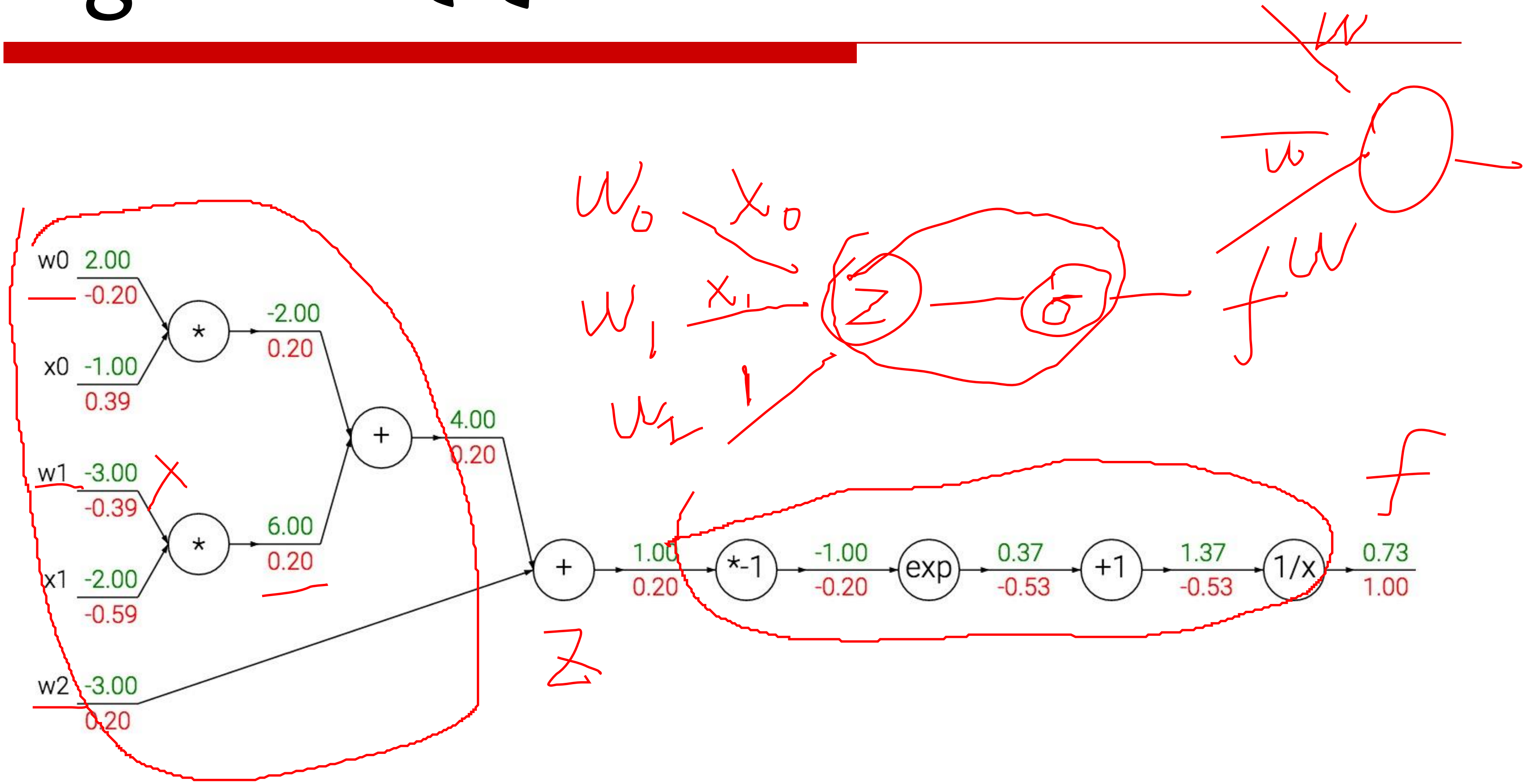
$-e^{-x}$

$\sigma' = \frac{-(\ )}{(\ )^2} =$

Sigmoid例子

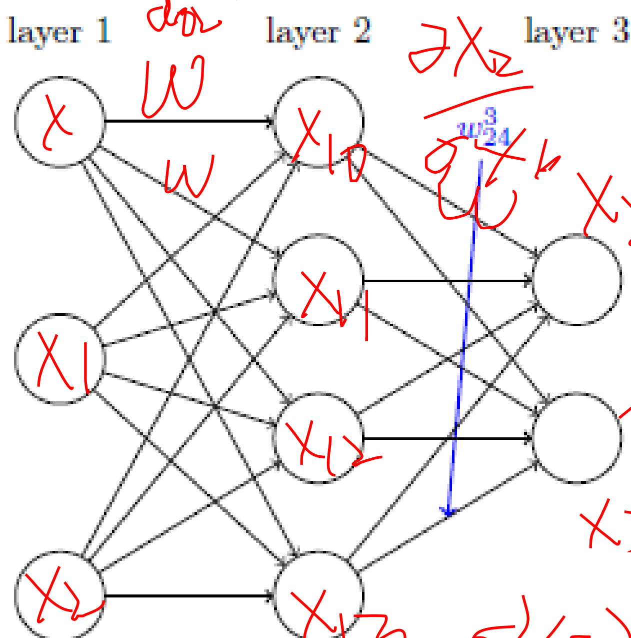
```
w = [2,-3,-3] # 我们随机给定一组权重
x = [-1, -2]

# 前向传播
dot = w[0]*x[0] + w[1]*x[1] + w[2]
f = 1.0 / (1 + math.exp(-dot)) # sigmoid函数

# 反向传播经过该sigmoid神经元
ddot = (1 - f) * f # sigmoid函数偏导
dx = [w[0] * ddot, w[1] * ddot] # 在x这条路径上的反向传播
dw = [x[0] * ddot, x[1] * ddot, 1.0 * ddot] # 在w这条路径上的反向传播
# yes! 就酱紫算完了! 是不是很简单?
```

Sigmoid例子


w0 2.00
-0.20
x0 -1.00
0.39
w1 -3.00
-0.39
x1 -2.00
-0.59
w2 -3.00
0.20
*
-2.00
0.20
*
6.00
0.20
+
4.00
0.20
+
1.00
0.20
*-1
-1.00
-0.20
exp
0.37
-0.53
+1
1.37
-0.53
1/x
0.73
1.00

Sigmoid神经网络的例子

$$\frac{\partial L}{\partial z_1} = L\left(\frac{\partial L}{\partial z_1}\right)$$



w^l_{jk} 是从 $(l-1)^{th}$ 层的第 k^{th} 个神经元到 l^{th} 层的第 j^{th} 个神经元的连接上的权重

$$\frac{\partial L}{\partial w^1} = \frac{\partial z}{\partial w^1}\frac{\partial L}{\partial z_2}$$

$$\frac{\partial L}{\partial z_2} = \frac{\partial a_{10}}{\partial z_2}\,\sigma'(a_0)\,\frac{\partial z_1}{\partial a_{10}}\,w_1\,\frac{\partial L}{\partial z_1}$$

Sigmoid神经网络的例子

$$a_j^l = \sigma\left(\sum_k w_{jk}^l a_k^{l-1} + b_j^l\right)$$

$C = C0, b$

$$a^l = \sigma(w^l a^{l-1} + b^l)$$

z

$W^T \cdot \vec{a} = Z$

$$z_j^l = \sum_k w_{jk}^l a_k^{l-1} + b_j^l;$$

$a_i \quad a_{i+1} = \sigma(z)$

$$\delta_j^l \equiv \frac{\partial C}{\partial z_j^l}$$

Sigmoid神经网络的例子

总结：反向传播的四个方程式

$$\frac{\partial L}{\partial z^L} = \delta^L = \underline{\nabla_a} C \odot \sigma'(z^L)$$

$$a = \sigma(z)$$

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$$

$$\frac{\partial C}{\partial b^l_j} = \delta^l_j$$

$$\frac{\partial C}{\partial w^l_{jk}} = \underline{a^{l-1}_k \delta^l_j}$$

$$\frac{\partial L}{\partial W} = \frac{\partial z}{\partial W} \times \frac{\partial L}{\partial z}, \quad z = a^{l-1}$$

a

Sigmoid神经网络的例子

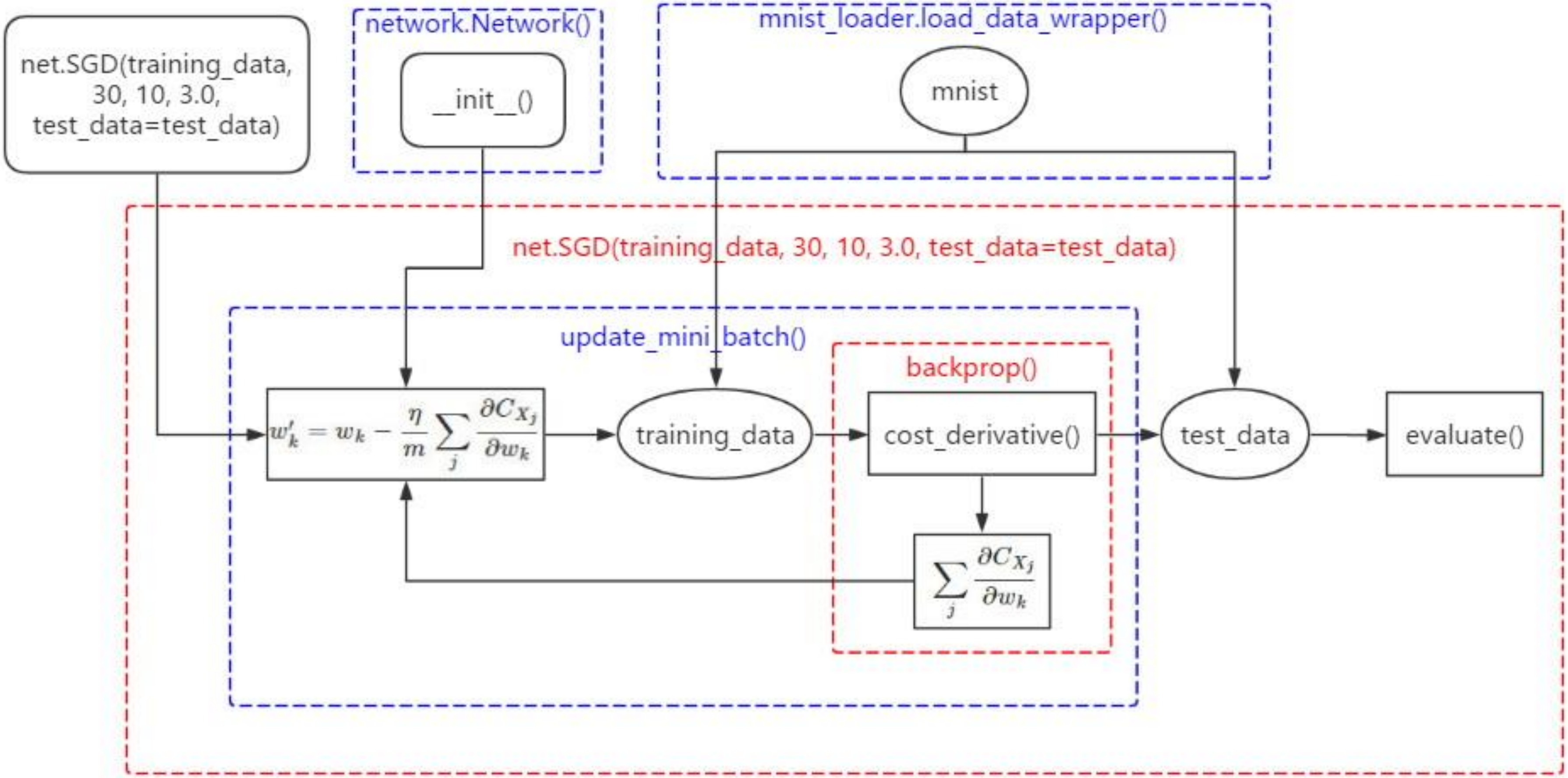
1. **输入 x：** 为输入层设置对应的激活值 a^1 。
2. **前向传播：** 对每个 $l = 2, 3, ..., L$ 计算相应的 $z^l = w^l a^{l-1} + b^l$ 和 $a^l = \sigma(z^l)$
3. **输出层误差 δ^L：** 计算向量 $\delta^L = \nabla_a C \odot \sigma'(z^L)$

∂l

Zout

4. **反向误差传播：** 对每个 $l = L-1, L-2, ..., 2$，计算 $\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$
5. **输出：** 代价函数的梯度由 $\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j$ 和 $\frac{\partial C}{\partial b^l_j} = \delta^l_j$ 得出

74行代码实现手写数字识别



感谢大家！

恳请大家批评指正！