CS231n学习笔记

Assignment Git:

<https://github.com/CS231n-zju/CS231n>

视频地址：

<https://www.youtube.com/watch?v=vT1JzLTH4G4&index=2&list=PLe7764SJVnV10-Nr7e0sBlC9J0LRf4sQo>

课程作业：

<http://cs231n.github.io/>

Syllabus:

<http://cs231n.stanford.edu/syllabus.html>

Note翻译:

<http://www.52ml.net/17723.html>

**Week2**

[Optimization: Stochastic Gradient Descent 2](#_Toc496656639)

[Visualizing the loss function 2](#_Toc1206787142)

[Optimization 2](#_Toc1576983284)

[Strategy #1: A first very bad idea solution: Random search 2](#_Toc1422764342)

[Strategy #2: Random Local Search 2](#_Toc12674405)

[Strategy #3: Following the Gradient 2](#_Toc235631072)

[Computing the gradient 3](#_Toc48960178)

[Computing the gradient numerically with finite differences 3](#_Toc640517340)

[Computing the gradient analytically with Calculus 4](#_Toc575762070)

[Gradient Descent 4](#_Toc385787414)

[Unknown word 4](#_Toc501704426)

[Video3.Lecture 3 | Loss Functions and Optimization 5](#_Toc1221782232)

[Unknown word 6](#_Toc1971176764)

[Assignment1\_Q2: Training a Support Vector Machine 6](#_Toc1676285614)

[1.Numpy.reshape 6](#_Toc388681015)

[2.Np.hstack() 7](#_Toc173139225)

[3.Why we have to subtract mean image from the training dataset? 7](#_Toc629803921)

[4.计算SVM的analytic梯度 7](#_Toc878262990)

[5.assignment回顾 7](#_Toc1392744842)

[Unknown word 8](#_Toc570889522)

[Backpropagation, Intuitions 8](#_Toc990255775)

[Simple expressions and interpretation of the gradient 8](#_Toc275508697)

[Compound expressions with chain rule 8](#_Toc1440834523)

[Modularity: Sigmoid example 9](#_Toc1393790753)

[Backprop in practice: Staged computation 9](#_Toc2137692591)

[Patterns in backward flow 9](#_Toc343104141)

[Summary 10](#_Toc1626498292)

[Unknown word 10](#_Toc1501208150)

[Linear backprop example 10](#_Toc1846373589)

[Unknown word 10](#_Toc174351633)

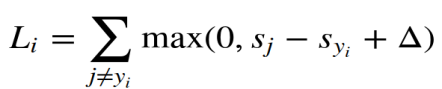
[Q&A 10](#_Toc604401944)

# Optimization: Stochastic Gradient Descent

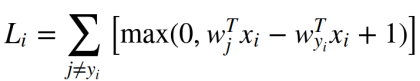
Optimization is the process of finding the set of parameters *W*W that minimize the loss function.

## Visualizing the loss function

从上一个chapter,得到loss function如下:



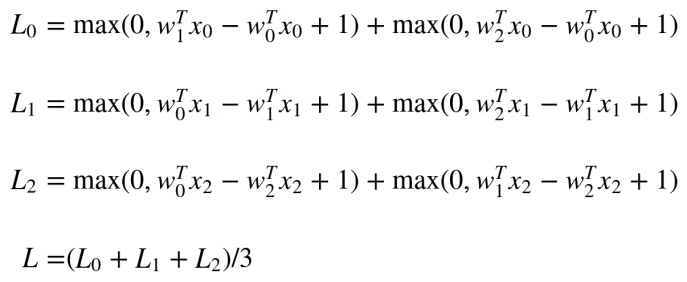
换一种写法:



其中wj是类j的权重向量.可以发现,L其实是wj的一个线性函数的和.

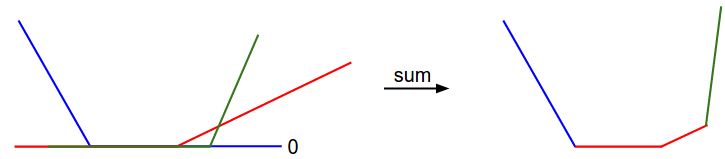
假设现在每张图xi只有一个维度,给三个图,他们的ground truth分别是class1,2,3.

那么,图0,1,2(x0,x1,x2)的loss分别如下:



下图是以w0为例的loss的可视化.左边是L0,L1,L2,右边则是L.

其中横轴为w0.



可以发现,sum of loss是一个碗装的凸函数,我们可以优化该函数得到最优解.

上图中x是一维的,如果变成多维状态,sum of loss将不再是一个凸函数.

注意到loss function是不可微分的,由于max函数,导致函数中存在一定的不可导点.

## Optimization

### Strategy #1: A first very bad idea solution: Random search

就是随机选择weights,然后看哪个loss低.

Our strategy will be to start with random weights and iteratively refine them over time to get lower loss.

有一种方式是把这个过程看作是盲人登山,山上每一个点都是一个loss(山的海拔).

### Strategy #2: Random Local Search

随机选择一个方向,随机走一个step\_size,如果loss下降,则继续往下走.

Ps: np.random.randn(10, 3073) \* 0.001

原型numpy.random.randn(d0, d1, ..., dn)

其中d0, d1, ..., dn : int, optional.The dimensions of the returned array, should be all positive.

### Strategy #3: Following the Gradient

不需要随机选择方向,而是可以用数学方式计算出最速下降的方向.

The gradient is just a vector of slopes (more commonly referred to as derivatives) for each dimension in the input space.

## Computing the gradient

两种方式:缓慢,近似但非常简单的数值梯度;快速但是易错,需要微积分的解析梯度.

### Computing the gradient numerically with finite differences

这部分代码使用迭代器:

**def** **eval\_numerical\_gradient**(f, x):

"""

a naive implementation of numerical gradient of f at x

- f should be a function that takes a single argument

- x is the point (numpy array) to evaluate the gradient at

"""

fx **=** f(x) *# evaluate function value at original point*

grad **=** np**.**zeros(x**.**shape)

h **=** 0.00001

*# np.nditer原来是numpy array自带的迭代器*

*#flags=['multi\_index']表示对a进行多重索引.*

*#op\_flags=['readwrite']表示不仅可以对a进行read（读取）,还可以write（写入）.*

*即相当于在创建这个迭代器的时候，我们就规定好了有哪些权限。*

it **=** np**.**nditer(x, flags**=**['multi\_index'], op\_flags**=**['readwrite'])

**while** **not** it**.**finished:

*# evaluate function at x+h*

*#it.multi\_index表示元素的索引*

ix **=** it**.**multi\_index

old\_value **=** x[ix]

x[ix] **=** old\_value **+** h *# increment by h*

fxh **=** f(x) *# evalute f(x + h)*

x[ix] **=** old\_value *# restore to previous value (very important!)*

*# compute the partial derivative*

grad[ix] **=** (fxh **-** fx) **/** h *# the slope*

*#表示进入下一次迭代*

it**.**iternext() *# step to next dimension*

**return** grad

这里h虽然在数学意义上趋向于0,但是这里用一个极小值1e-5代替就可以了.

新的weight向着df的反方向走(df就是grad).

W\_new **=** W **-** step\_size **\*** df

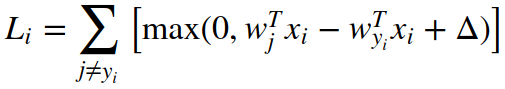
Step\_size就是learning rate,取值过大可能progress很大但是会有risky.取值过小progress过小.

用这种方式,对于有10\*3072个参数的网络而言,我们每走一步,要计算30721个loss,这对于现代神经网络所拥有的庞大参数量而言,非常worse.

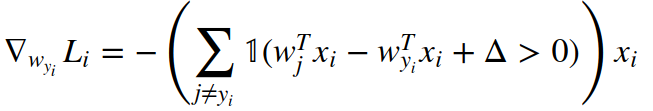
### Computing the gradient analytically with Calculus

analytic gradient是易错的,因此常常需要用的numerical gradient和它进行check.

SVM的loss function如下:



现在我们要求Loss对于w的微分,对于每一张图,有class行w,对于每一行w,如果是ground truth那一行,就是:



Ps:这里1指的是,括号里面的判别式取true则为1,否则取0.

如果不是,则是:

2017-10-29 00-47-49屏幕截图

## Gradient Descent

优化的核心:

*# Vanilla Gradient Descent*

**while** True:

weights\_grad **=** evaluate\_gradient(loss\_fun, data, weights)

weights **+=** **-** step\_size **\*** weights\_grad *# perform parameter update*

由于计算所有的gradient会非常的wasteful,所以我们选择一个小batch做这一工作(也会更加容易收敛).

*# Vanilla Minibatch Gradient Descent*

**while** True:

data\_batch **=** sample\_training\_data(data, 256) *# sample 256 examples*

weights\_grad **=** evaluate\_gradient(loss\_fun, data\_batch, weights)

weights **+=** **-** step\_size **\*** weights\_grad *# perform parameter update*

最极端的是batch\_size=1的情况,被称为随机梯度下降(on-line gradient descent). batch\_size的大小一般不用cross-validation来确定.

## Unknown word

march along 行进

piecewise-linear 分段线性

bumpy  颠簸的

Interchangeably 互换的

To reiterate, 重申

Perturbations 扰动

steepest descend 最速下降

at least in the limit as the step size goes towards zero 至少在步长接近零的范围内

instantaneous rate 顺势速率

error-prone 易错的

Calculus 微积分

generic function 通用函数

headache-inducing 令人头疼的

overshoot 超调

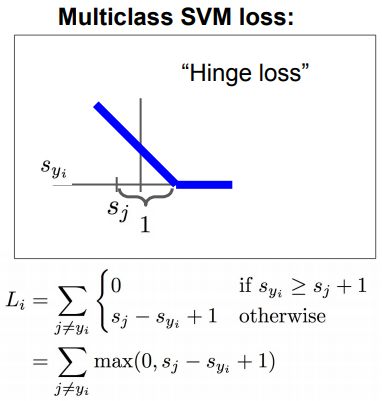
Derive 得到

vanilla 香草

Stochastic 随机的

# Video3.Lecture 3 | Loss Functions and Optimization

SVM loss.横轴为syi的值,可以发现,当syi比sj多的部分超过margin(here setting the margin equals to 1), 那么loss will always been zero.



Maximum of loss is infinity and the minimum is zero.

Trick:如果初始时所有的w都是0,那么score一直是0,loss是多少呢?

如果把delta设置为1的话,loss是class的个数,可以用来检测程序初始的时候有没有什么bug.

将loss中的max换成mean不会改变最终的loss.

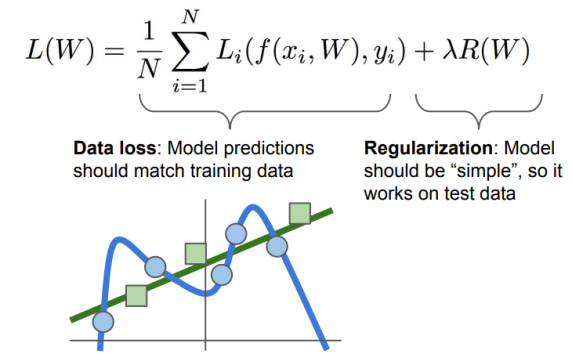
正则项的真正意义!

未添加正则项的时候,loss会使得w的表达更拟合training data,但是添加正则项之后,可以使得w更拟合unseen data in the future.

More specifically, 正则项使得拟合曲线更偏向于更低的degree.如下图,为了更好地拟合training data(蓝色点),我们得到蓝色曲线,然而对于test data,他显然是不适用的.

一种说法是,我们更倾向于选择低次的表达(simplest one).

正则项真正在做的是,如果你要用更复杂的表达,那么你必须overcome这个正则penalty,他对于更高次的表达,惩罚越高.



对于softmax,一种debug方法是:

如果把w全都是0,那么score都是0,那么exp(0)=1,因此初始时loss是-log(1/C)

## Unknown word

turn in 上交

hone in 转向

Cetera 等等

Mindset 思维观念

Diagram 图表

Unravel 解开

Intensity 强度

left off 离开

Compactify 紧致化

Incur 招致

screw up 搞砸了

Offend 冒犯

aesthetic sensibility 美感

scaling up 放大

Derivation 推导

Faintly 微弱的

Jiggle 微动

by convention 依照惯例

ahead of time 提前的

weigh off 权衡

Catastrophic 灾难性的

Versus 与...相比

Weired 奇怪的

canonical example 典型的例子

Astray 误入歧途

polynomial basis function 多项式基函数

coarse 粗糙的

Inference 推断

Likehood 似然

sanity check 完整性检查

spoiler alert 剧透

sheer dumb luck 纯粹依靠运气

Scalar 标量

Incorporate 结合

Diagonal 对角线

Teaser 戏弄者

Tying 搭

# Assignment1\_Q2: Training a Support Vector Machine

## 1.Numpy.reshape

Numpy.reshape(a, newshape, order='C')

newshape : One shape dimension can be -1. In this case, the value is inferred from the length of the array and remaining dimensions.

## 2.Np.hstack()

Np.vstack()

H---horizontal 按列组合

V---vertical 按行组合

## 3.Why we have to subtract mean image from the training dataset?

原先图像中每一个通道的数值在:[0…255]范围内,去均值化导致taeny分布在[-127 … 127]之内.去均值的目的是归一化,归一化有很多优势:

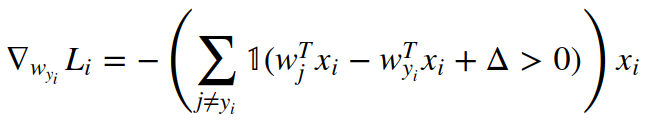
<https://zhuanlan.zhihu.com/p/30358160>

(1)加快梯度下降求解速度

(2)提升精度

## 4.计算SVM的analytic梯度

See in optimization:

yi那一行

非yi那一行2017-10-29 00-47-49屏幕截图

“when you’re implementing this in code you’d simply count the number of classes that didn’t meet the desired margin (and hence contributed to the loss function) and then the data vector *xi* scaled by this number is the gradient.”

## 5.assignment回顾

(1)linear\_svm.py中loss和gradient的实现

ps:注意向量化写法的实现. linear\_svm.py : svm\_loss\_vectorized(W, X, y, reg)

1. linear\_classifier.py中train()和predict()的部分

Train的过程:

每次随机选取training data中的batch\_size这么多的data,将这个过程迭代num\_iters次.即weight沿着loss下降的方向走num\_iters次.

每次计算gradient的时候对batch\_size这么多data导致的loss和gradient求均值.

(3)mySVM.py中用validation data去fine tune一些超参数(Q2里面是learning rate和regularization strength).

用training data和train方法训练出一个svm,去predict validation data, 找到最好的一个超参数组合,他们的accuracy最高.

## Unknown word

Compatible 相容的

sanity check 完整性检查

# Backpropagation, Intuitions

Backpropagation: computing gradients of expressions through recursive application of chain rule.

## Simple expressions and interpretation of the gradient

主要介绍了一些求导的方法和意义.

## Compound expressions with chain rule

The **chain rule** tells us that the correct way to “chain” these gradient expressions together is through multiplication.

通过连续乘法计算链式梯度.

BP的visualization举例:

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

q **=** x **+** y

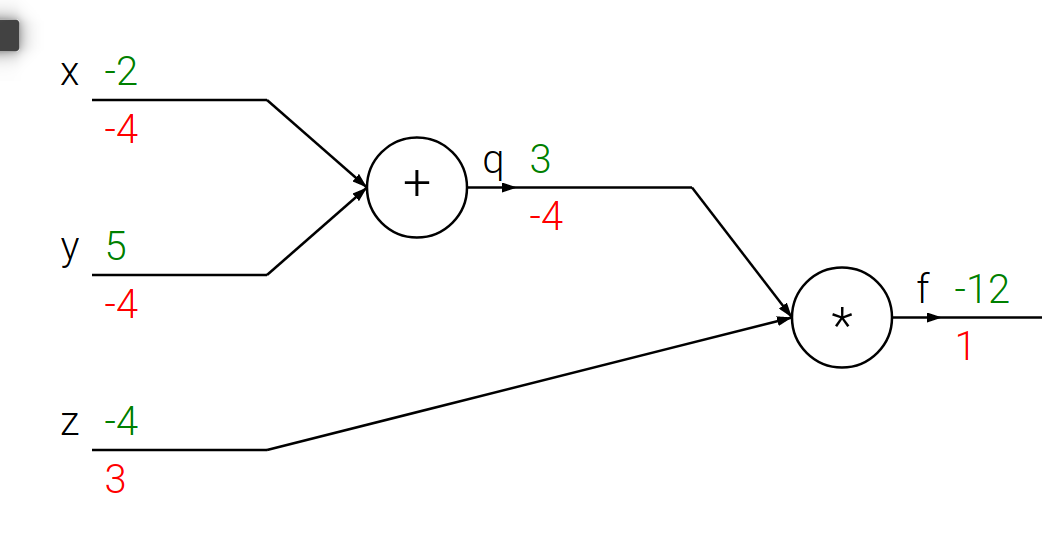
f **=** q **\*** z

dfdz **=** q

dfdq **=** z

dfdx **=** 1.0 **\*** dfdq

dfdy **=** 1.0 **\*** dfdq



Every gate compute two things:

1. its output value

2. the local gradient of its inputs with respect to its output value.

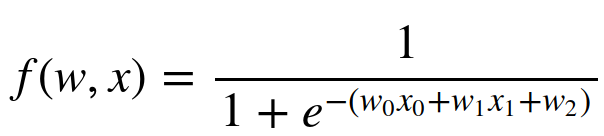
Gate的一些意义,以输出q的加法门为例,gradient为负(-4),表示如果我们希望最终的output升高,那么希望下次从这个”+”gate输出的值更小一些.

事实确实如此,如果x,y减小,加法门的输出将会减小,同时f的输出却会增大.

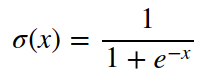
Backpropagation can thus be thought of as gates communicating to each other (through the gradient signal) whether they want their outputs to increase or decrease (and how strongly), so as to make the final output value higher.

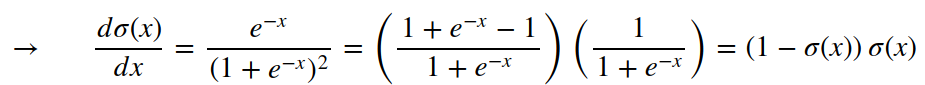
所以BP可以看做是gate之间通过gradient信号互相通信.为了最终的output提升,gate希望自己的输出高一些还是低一些(取决于梯度信号的正和负).

## Modularity: Sigmoid example



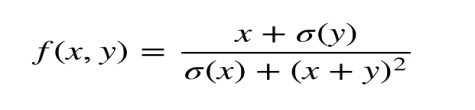
利用求导过程中先+1后-1的方法,可以简化导数:

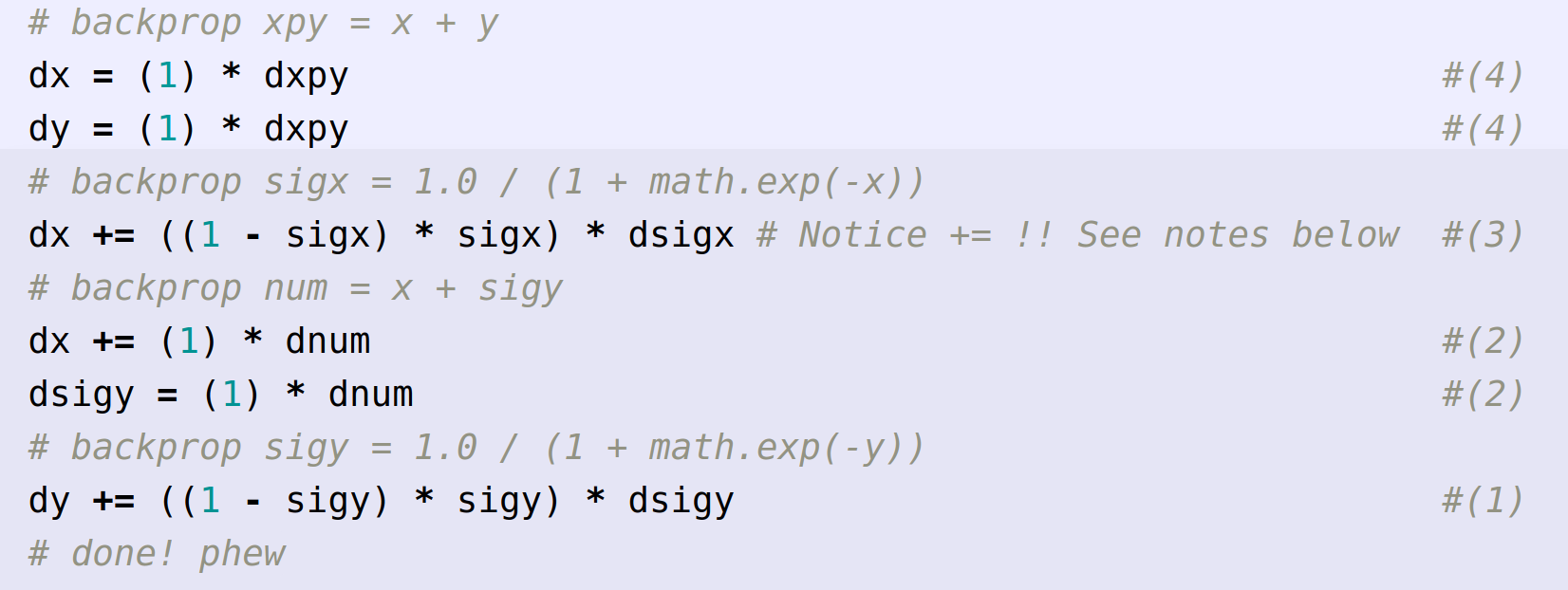




## Backprop in practice: Staged computation

计算下式中的dx时,代码中用了+=,是梯度累加!





根据链式法则: if a variable(这里是x) branches out to different parts of the circuit, then the gradients that flow back to it will add.

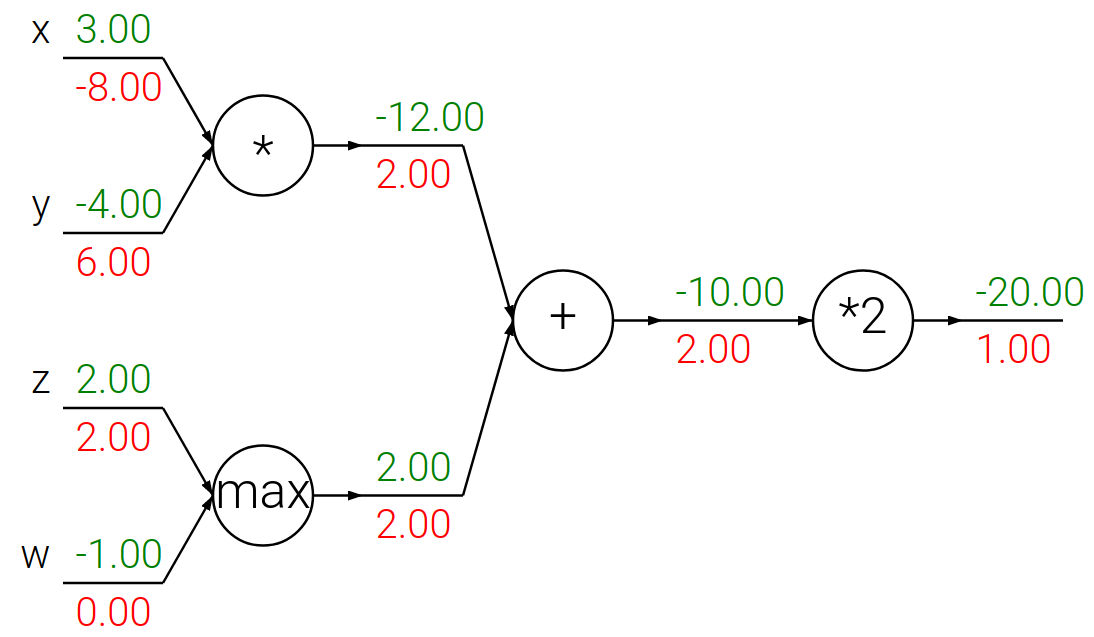
## Patterns in backward flow

以下图为例,可以发现,加法乘法max三种门,对于他们的输出梯度(门右边的),如何将这个梯度分发给输入.

加法:相等地分发.对于2x+3y,2x和3y的系数都是1.

max:取较大的.对于max(x,y),如果取了x那么x的贡献就是1,y就是0.

乘法比较复杂:相乘的几项中,较大的输入项,会给一个较小的梯度,反之亦然.



## Summary

这章主要是介绍怎么求导数的,注意分段求导的使用,毕竟我们不需要用一个超级大的公式来表达导数.

## Unknown word

Subtleties 微妙/巧妙之处

Notation 记号

Analogously 类似的

Cog 小成分

Unary 一元的

Simplifies 简化

Numerator 分子

Ultimately 最终

Staged computation 分段计算

Denominator 分母

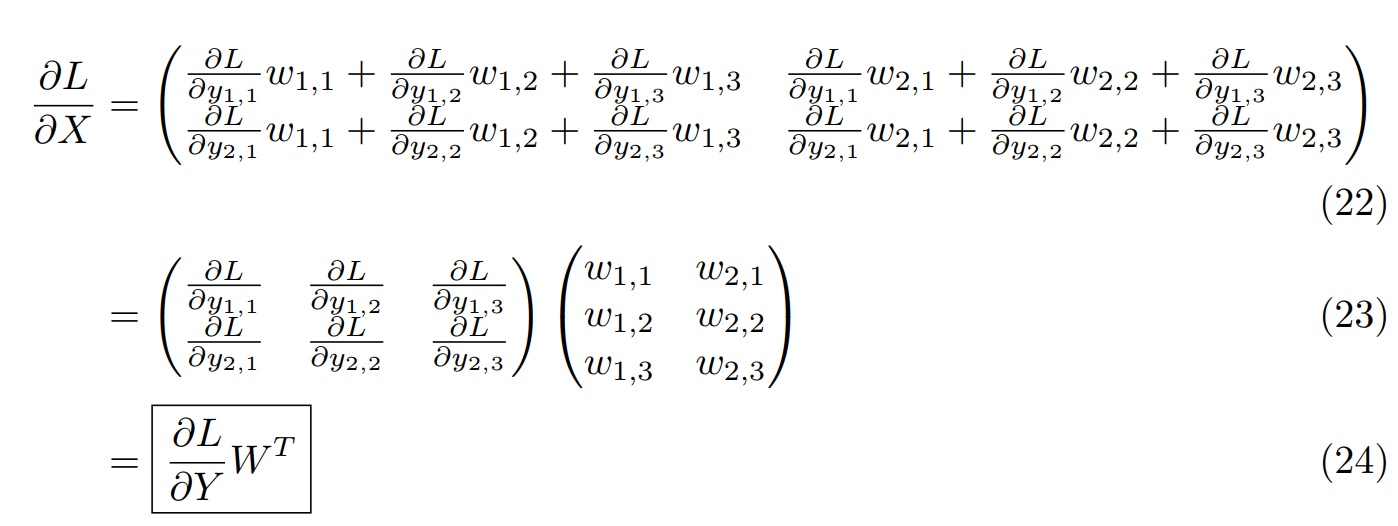
Circuit 电路

straight-forward manner 直截了当的方式

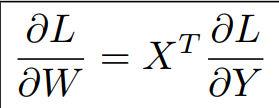
Transpose 转置

# Linear backprop example

计算loss对于x的梯度公式的推导,最终得到:



同理,loss对W的梯度公式:



这里解释了Q2和Q3里求dW公式的向量化算法.

## Unknown word

Scalar 标量

Solely 唯一的

# Q&A

Optimization:

Visualizing the loss function 解释没看懂

Computing the gradient analytically with Calculus这里j=yi和j不等于yi的两个梯度含义没有看懂

Video3 22分钟

为什么改变成mean, loss不会变?