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>

**Week3**

[Video4.Lecture 4 | Introduction to Neural Networks 2](#_Toc556947373)

[Unknown word 3](#_Toc1681258820)

[Assignment1\_Q3: Implement a Softmax classifier 4](#_Toc81250404)

[1.计算softmax的loss和gradient 4](#_Toc100220741)

[2.assignment回顾 4](#_Toc506548262)

[Neural Networks Part 1: Setting up the Architecture 4](#_Toc328361428)

[Single neuron as a linear classifier 5](#_Toc1190755591)

[几个激活函数 5](#_Toc1519021606)

[神经网络结构 5](#_Toc773811659)

[神经网络的表达能力 6](#_Toc1225273648)

[Unknown word 6](#_Toc399823729)

[Neural Networks Part 2: Setting up the Data and the Loss 7](#_Toc607568260)

[数据预处理 7](#_Toc1466418271)

[权值初始化 8](#_Toc1280944841)

[正则化 8](#_Toc1539987783)

[损失函数 9](#_Toc616484402)

[分类问题 9](#_Toc1544366109)

[回归问题 9](#_Toc423612650)

[结构化预测 9](#_Toc109810065)

[Summary 10](#_Toc1374031929)

[Unknown word 10](#_Toc656946915)

[Neural Networks Part 3: Learning and Evaluation 10](#_Toc616064341)

[梯度检查 10](#_Toc529207831)

[一些明智的检查trick 11](#_Toc552953414)

[学习过程 11](#_Toc704046249)

[(1)loss 11](#_Toc1147674779)

[(2)train/validate accuracy 11](#_Toc1661126628)

[(3)更新率 12](#_Toc361118661)

[(4)每层的激活值 12](#_Toc1342608687)

[(5)可视化 12](#_Toc585907792)

[参数更新 12](#_Toc360781543)

[普通更新 12](#_Toc1899556060)

[动量更新 12](#_Toc119682964)

[Nesterov动量 12](#_Toc442031947)

[退火学习率下降 12](#_Toc1999776801)

[逐参数适应学习率 12](#_Toc626231227)

[1.Adagrad 13](#_Toc770393375)

[2.RMSprop 13](#_Toc1043048744)

[3.Adam 13](#_Toc2145252833)

[超参数调优 13](#_Toc1544205034)

[1.实现问题 13](#_Toc120838745)

[2.用一个验证集而不是交叉验证 13](#_Toc397592915)

[3.超参数范围 13](#_Toc4289646)

[4.随机搜索比网格搜索结果更好 13](#_Toc1587257016)

[5. 边界最优值 13](#_Toc1678537756)

[6. 从粗糙到细致地搜索 13](#_Toc1544277429)

[模型集成 13](#_Toc56257771)

[总结 14](#_Toc1075420217)

[Unknown word 14](#_Toc1967890079)

[Putting it together: Minimal Neural Network Case Study 14](#_Toc166067836)

[计算loss 14](#_Toc301968499)

[计算gradient 15](#_Toc477353346)

[Unknown word 15](#_Toc782132178)

[Assignment1\_Q4: Two-Layer Neural Network 15](#_Toc831176330)

[1. 一些问题 15](#_Toc1030306760)

[2.我的方法(my\_two\_layer\_net.ipynb和my\_neural\_net.py) 15](#_Toc1486178427)

[2.1 forward问题 15](#_Toc1978851109)

[2.2 Backward 15](#_Toc543949740)

[3.根据官方note完成 16](#_Toc1847297089)

[3.1 forward 16](#_Toc1173976149)

[3.2 backward 16](#_Toc1129857532)

[Unknown word 17](#_Toc60594984)

[Assignment1\_Q5: Higher Level Representations: Image Features 17](#_Toc926048561)

[1.核心Extract Features 17](#_Toc1249540497)

[2. Bug 17](#_Toc502626931)

[Unknown word 17](#_Toc778341715)

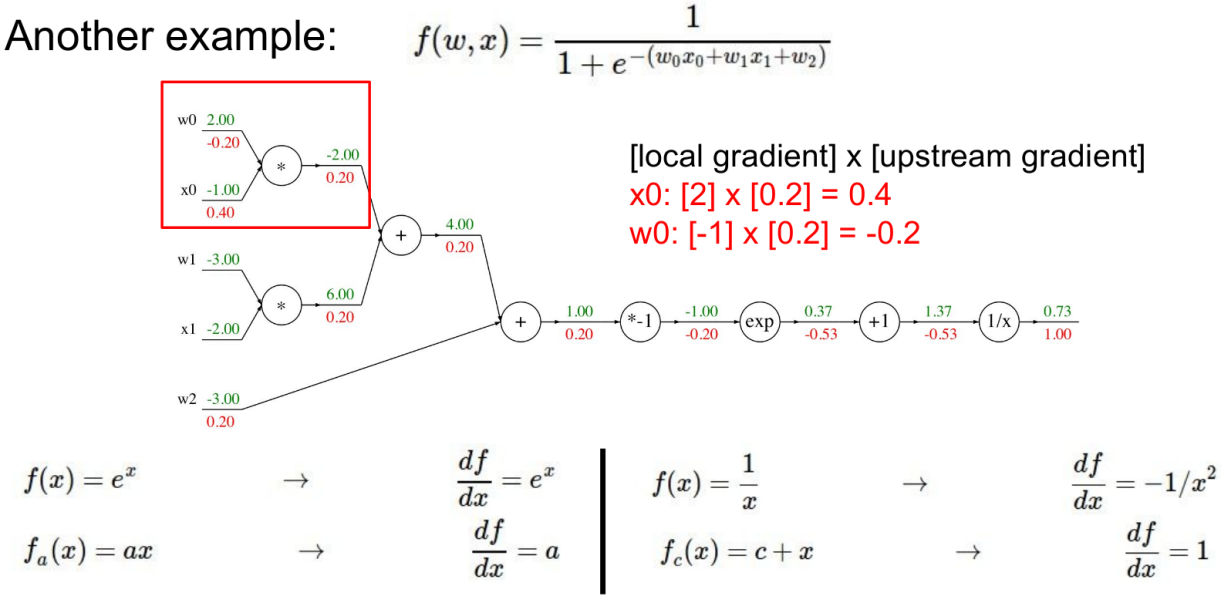
[Q&A 18](#_Toc1875771724)

# Video4.Lecture 4 | Introduction to Neural Networks

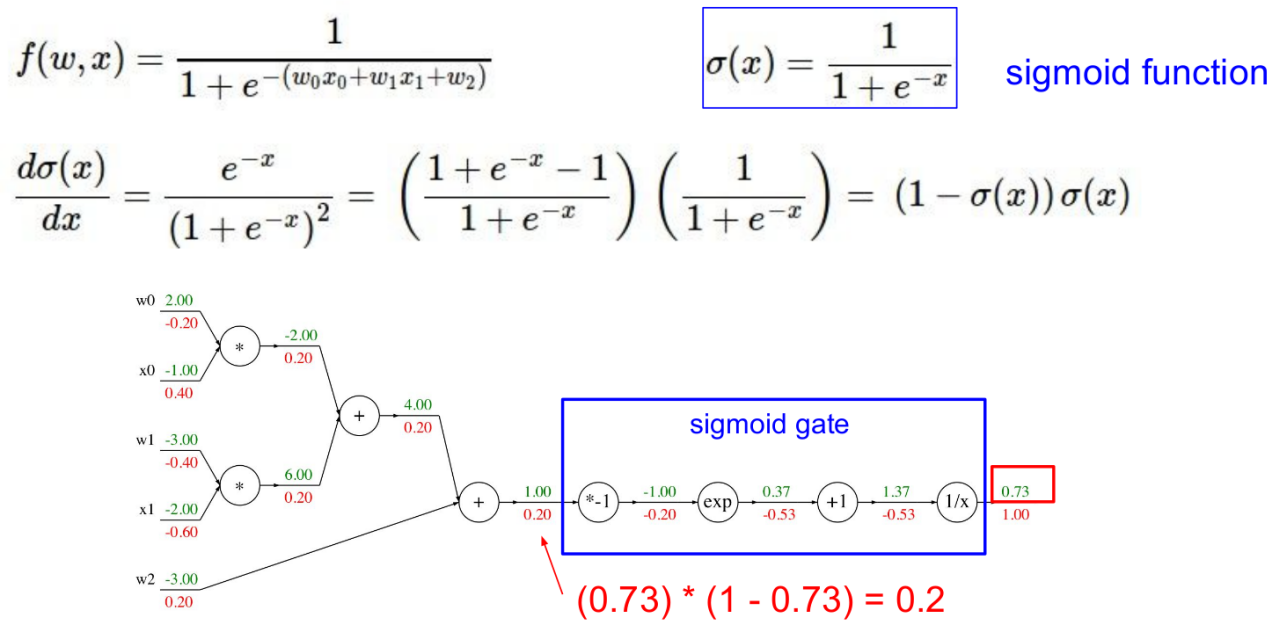
计算每一个反向传播的node的左边的 gradient的时候,值=local gradient\*upstream gradient.

local gradient也就是这个node对所求变量的partial derivative.

upstream gradient也就是往前传的上一步的gradient.



注意到如果把后面sigmoid的梯度求解部分变成一个整体,也可以得到0.2这个结果.



基于上文,可以发现三种gates: 和/乘/max分别有三种含义:

add gate: gradient distributor

max gate: gradient router

mul gate: gradient switcher

## Unknown word

Trajectory 弹道,轨迹

Insane 疯狂的

Unrolling 展开

Trivial 无价值的

plug in 插入

Granularity 粒度

Versus 对抗

diagonal matrix 对角矩阵

Stack linear layers 叠加线性层

Collapse 碰撞

Impulse 脉冲

Axon 轴突

Synapse 突触

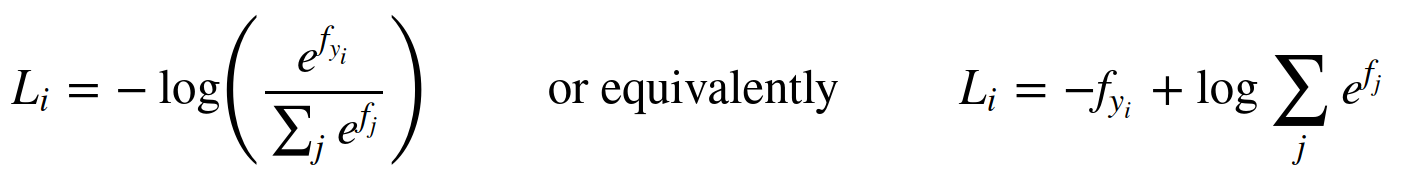
Dendrite 树突（位于神经元末端的细分支，接收其他神经元传来的信号）

draw analogy 做类比

# Assignment1\_Q3: Implement a Softmax classifier

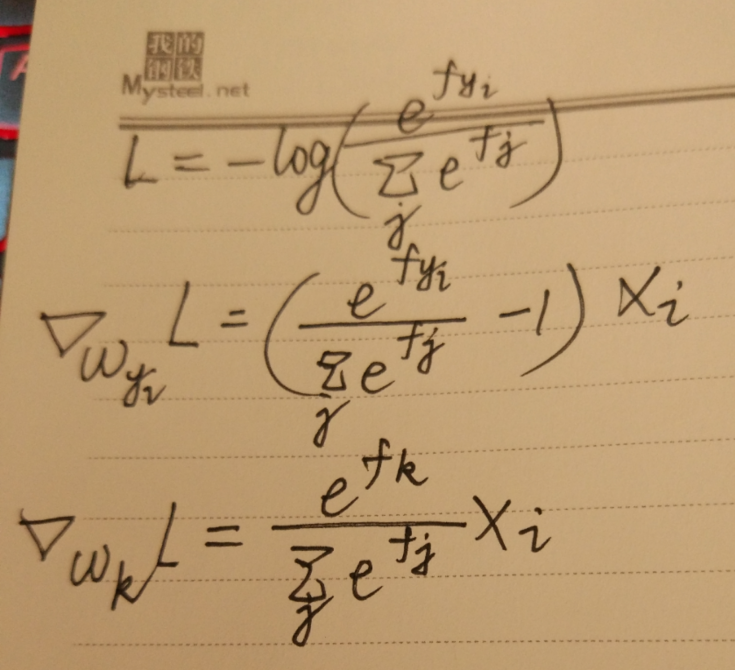
## 1.计算softmax的loss和gradient

Loss公式:



Gradient公式:

我自己的推导结果是



别人的推导过程:

<http://blog.csdn.net/yc461515457/article/details/51924604>

<http://blog.csdn.net/pjia_1008/article/details/66972060>

这两个版本对于j≠yi的推导都是错误的,分子应该是e^(f\_j)而不是e^(f\_(y\_i))

## 2.assignment回顾

(1)softmax.py中loss和gradient的实现

ps:注意向量化写法的实现.

1. mySoftmax.py

实现和svm完全一样

# [Neural Networks Part 1: Setting up the Architecture](http://cs231n.github.io/neural-networks-1/)

介绍一个简单的双层神经网络,输入X=[3072,1],w1=[100,3072],w2=[100,10]

w1\*X=[100,1]

Score=w2.T\*max(0,w1\*X)=[10,1]

## Single neuron as a linear classifier

通过loss,我们可以把一个神经元当做一个线性分类器:Softmax/SVM.

这里regularization interpretation就是逐渐遗忘,因为它的目的就是令所有的w越接近于0越好(越接近于0,loss=λ\*sum(Wi)越低)

## 几个激活函数

Sigmoid:输出[0,1].现在不常用了,主要是因为两个原因:

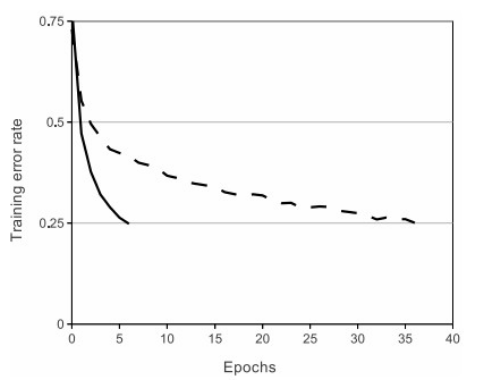
1. sigmoid在饱和的时候梯度很小.就是说它容易令激活值接近0或者1的位置的gradient接近于0,因为求gradient的时候总是需要用(local gradient) × (upstream gradient),如果local gradient很小那么网络再往前传播的时候很容易没有gradient了,网络也就不能学习.
2. sigmoid不是关于0对称的,会导致可能出现Z字形下降.

Ps:问题2相对于1不是那么严重

Tanh:输出[-1,1].用的很多.本质是sigmoid的一个放缩.

ReLu:输出max(0,x).

优点:①更快使得梯度下降收敛



②更容易实现.

缺点:learning rate很高的时候会容易杀死很多neurons.

为了解决提出了Leaky ReLU:

F(x)=1(x<0)

F(x)=ax+1(x>=0)

究竟用哪个!推荐用ReLu,如果恐惧神经元死亡问题,可以转而用Leaky ReLU来避免这种情况.

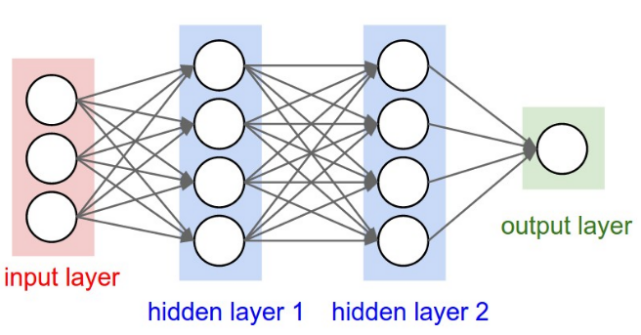
## 神经网络结构

命名层数问题:SVM被认为是单层神经网络的一种(input直接连接output).神经网络的层数不考虑input层.

Output层:往往没有激励函数,直接输出class score(分类问题),或者实值(回归问题).

神经网络的尺度:用参数量衡量(w和b).

以下图全连接网络结构为例的前向传播标程:



*# forward-pass of a 3-layer neural network:*

f **=** **lambda** x: 1.0**/**(1.0 **+** np**.**exp(**-**x))*# activation function (use sigmoid)*

x **=** np**.**random**.**randn(3, 1) *# random input vector of three numbers (3x1)*

h1 **=** f(np**.**dot(W1, x) **+** b1) *# calculate first hidden layer activations (4x1)*

h2 **=** f(np**.**dot(W2, h1) **+** b2) *# calculate second hidden layer activations (4x1)*

out **=** np**.**dot(W3, h2) **+** b3 *# output neuron (1x1)*

## 神经网络的表达能力

1.层数

2层全连接神经网络可以近似表达任何一个函数!但是为什么我们需要更深的网络呢?答:经验主义.

在全连接网络中3层会比2层表达效果更好,但是4,5......之后的深度对能力的提升作用不大.

在CNN中,深度却是一个非常重要的影响表达能力的因素!

1. 每层的个数

越多越好!

①越多越能拟合复杂的函数.好处是能拟合复杂的数据,坏处是容易过拟合.但是我们不会为了防止过拟合而降低神经元个数,其他方法可以用比如dropout...

②每层个数少的网络每次训练出来的结果经常是都不一样的,但是大一些的网络,每次的结果往往相似(loss差异不大).

## Unknown word

Wiggle 扭动

Stochastic 随机

Collapsed 瓦解

Nonetheless 虽然如此,但是

Billion 十亿

Synapses 突触

Dendrites 树突

Axon 轴突

Excitory 兴奋性

inhibitory 抑制

Spike 长钉,穗

Squash 挤压

Property 属性

Groaning 呻吟

Allude 提及

fully-saturated 完全饱和

Assumed 假定的

fallen out of favor 失去好感

Rectified 校正

Acyclic 无环

amorphous 无组织

Blob 一点

Pairwise 成对的

Interwoven 交织

Compactly 紧实的

Empirical 经验主义

Outperform 做得更好

Collaborate 合作

Conversely 反之

Reiterate 重申

Ubiquitous 无处不在的

# Neural Networks Part 2: Setting up the Data and the Loss

## 数据预处理

1.去均值:几何理解就是把每一个维度上的数据云中心都迁移到原点.

X -= np.mean(X)

1. 归一化(normalization):归一化这个流程只有当“不同特征的数据数据范围或者计量单位不同,但是他们对于算法具有相同的重要性”时才使用!

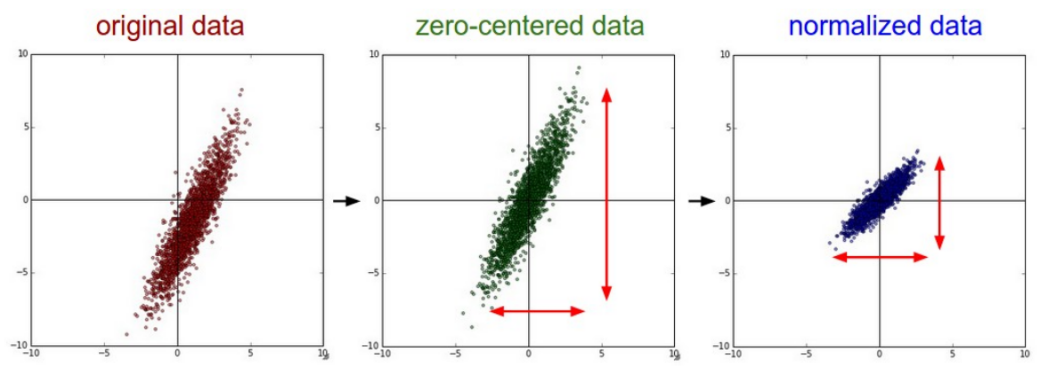
①先零中心化,再对每一个维度除以标准差.

X /= np.std(X, axis = 0

②对每一个维度归一化,这样min和max分别是-1和1

对于图片而言不需要做归一化了,因为他们的数据范围都是在[0,255]之间的.

完成1,2流程后的data:



3.PCA and Whitening

PCA是降维的一个过程,降维后可以保存最大方差,在提升训练模型表达能力的基础上,我们还可以节省时间和空间.

白化是把每一个维度除以它的eigenvalue,使得这个维度归一化.几何解释是if the input data is a multivariable gaussian, then the whitened data will be a gaussian with zero mean and identity covariance matrix.

实际应用中很少用PCA和白化但是零中心化和归一化还是很重要的.

注意:应该先把数据集划分成training/validation/test,然后计算训练集的均值,每个训练集都要减去这个均值,而不是各自的均值或所有数据的均值.

## 权值初始化

1.全是0是错误的!至少应该保证一半正数,一半负数.如果一开始所有的权值都相同的话,将会导致bp的时候梯度也都是一样的,失去了神经元之间不对称的源头.

2.初始化为较小的值来打破对称:

 W = 0.01\* np.random.randn(D,H)

Ps:并不是越小的权值就越好,它可能导致bp中返回很小的梯度,这在深度网络中并不是一件好事.

3.用n的平方根校准数据.

w = np.random.randn(n) / sqrt(n)

可以经验上使得数据聚集率提升,使得输入数据同分布.

对于ReLU神经元,可以这样初始化经验上效果更好:

  w = np.random.randn(n) \* sqrt(2.0/n)

1. 初始化偏置向量

初始化为0也可以,因为权值已经打破对称性了,对于ReLu神经元,有时候也都初始化为0.01,保证一开始就全部被激活(不能保证实验效果一定好,更多的时候是用全0).

实际应用中,用ReLU并且全部初始化为w = np.random.randn(n) \* sqrt(2.0/n)

5.批量归一化

在全连接层和非线性操作之间添加一个归一化层.

## 正则化

一些降低overfitting的方法.

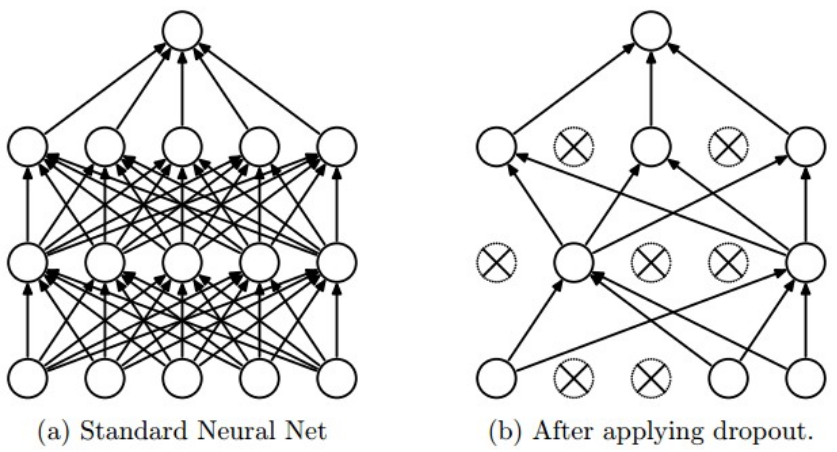
1.在输入激励函数的值后面加上L1/L2:2017-12-14 11-40-50屏幕截图,L2倾向于选择使得网络能够应用到全部输入数据而不是其中某一部分的权值(权值更弥散分布).如果没有明确的特征选择倾向,L2 outperform than L1.

2.最大范式约束(max norm constraint),防止learning rate过高时网络爆炸.

3.Dropout:

(1)原理:

训练过程中,only keeping a neuron active with some probability p (a hyperparameter), or setting it to zero otherwise.



如上图,只激活一部分神经元而不是整个网络.

测试过程并不进行dropout.

(2)实现:

在训练过程:

H1 **=** np**.**maximum(0, np**.**dot(W1, X) **+** b1)

U1 **=** np**.**random**.**rand(**\***H1**.**shape) **<** p *# first dropout mask*

H1 **\*=** U1 *# drop!*

在predict过程:

H1 **=** np**.**maximum(0, np**.**dot(W1, X) **+** b1) **\*** p *# NOTE: scale the activations*

使用inverted dropout来使得测试代码对训练代码untouched:

H1 **=** np**.**maximum(0, np**.**dot(W1, X) **+** b1)

U1 **=** (np**.**random**.**rand(**\***H1**.**shape) **<** p) **/** p *# first dropout mask. Notice /p!*

H1 **\*=** U1 *# drop!*

实际中,用全局L2正则,并且在每层后面加一个dropout也很常见,p值可以通过交叉验证取得.

## 损失函数

### 分类问题

1. 分类个数很多问题:分层softmax,用树形结构表达classifier,一条路径就是一个class.
2. 属性可选问题.有的class包含这个属性,但是有的并不包含.

①对于每个attribute定义一个classifier: 1选,0不选

② 对于每个attribute定义一个逻辑斯特回归函数: >0.5选,otherwise不选

### 回归问题

尽量用分类,分类相对于回归更能给出一个结果的分布,而不仅仅是一个精确值.

如果需要做回归,用L2 loss.

### 结构化预测

指的是预测结果较为复杂,不仅仅是一个label,可能包含树形结构等等,超范围.

## Summary

数据预处理:去均值化,归一化.

初始化权值: w = np.random.randn(n) \* sqrt(2.0/n).

Use L2 regularization and dropout.

Use batch normalization.

一些常见实际问题和针对这些问题使用的loss function.

## Unknown word

Involve 涉及

Symmetric 对称

covariance matrix 协方差矩阵

Eigenvector 特征向量

Decorrelate 去相关

Pitfall 陷阱

Reasonable 合理的

Asymmetry 不对称的

Calibrating 校准

Heuristic 启发式的

Propagate 传播

Alleviates 缓解

Intriguing 有趣的

Constraint 约束

Ensemble 集成

Attenuation 衰减

Augmentation 增强

Likelihood 可能性

Optimal 最佳的

monotonic 单调的

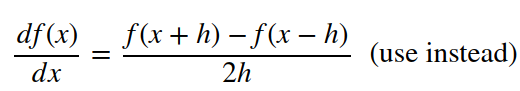
# Neural Networks Part 3: Learning and Evaluation

The process of learning the parameters and finding good hyperparameters.

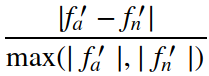
## 梯度检查

(1)比较计算出的和理论梯度.

理论梯度公式:



1. 使用相对误差而不是绝对误差:



为了防止除以0,会把两个式子相加.

判断标准:

|  |
| --- |
| relative error > 1e-2 usually means the gradient is probably wrong  1e-2 > relative error > 1e-4 should make you feel uncomfortable  1e-4 > relative error is usually okay for objectives with kinks. But if there are no kinks (e.g. use of tanh nonlinearities and softmax), then 1e-4 is too high.  1e-7 and less you should be happy. |

如果网络很深,有时候1e-2也可以被接受.

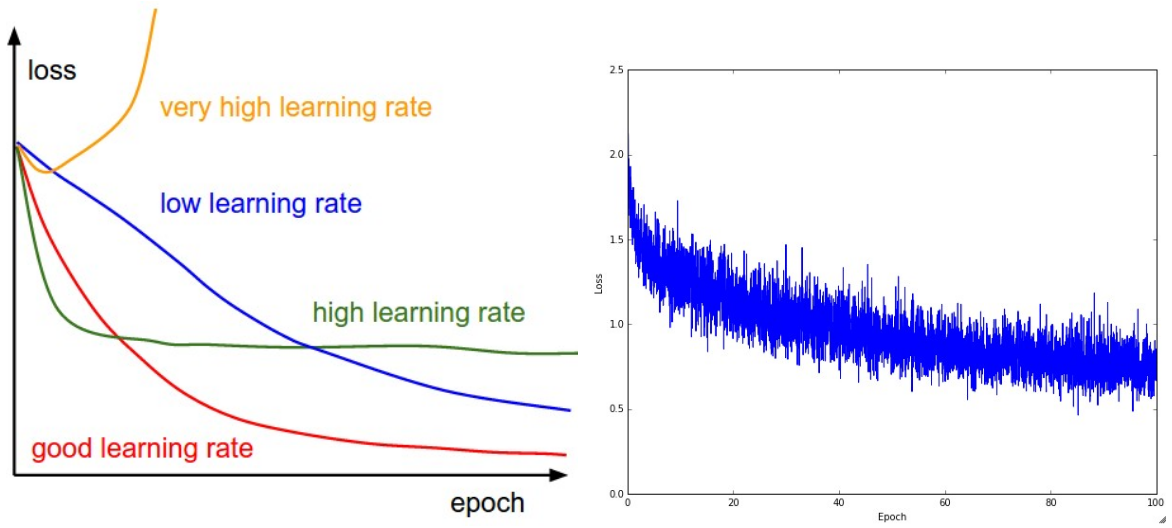
1. 精度问题:使用双精度来保存误差,防止出现截断.防止两个梯度都很小,最好打印出来看看.
2. 不可导点:有时候f(x+h)和f(x-h)这一步可能越过不可导点.通过跟踪max()函数中的赢家什么时候变化了可以认为越过了不可导点.
3. 使用少量数据做梯度检查.可以避免越过不可导点并且可以以偏概全,更快效率更高.
4. 步长并不是越小越好,有时候h过小会导致问题(3),一般1e-4~1e-6很合适.
5. 在网络预热一段时间,开始梯度下降之后再进行梯度检查.
6. 预防正则项overwhelm数据.可以先去掉正则项或者直接加强正则项.
7. Check的时候去掉dropout这些.有可能导致dropout未经检查有错误产生,可以通过计算f(x+h)和f(x-h)前添加随机种子.

## 一些明智的检查trick

1. 检查特定值处的loss.
2. 随着正则化项加强,loss应当增加.
3. 先过拟合小数据集.

## 学习过程

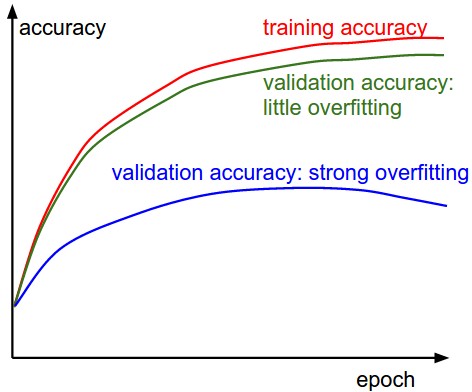
### (1)loss



非常低的learning rate可能会导致下降曲线近似线性,右图为正常的下降曲线.

### (2)train/validate accuracy

相差过大说明过拟合,如影随形说明模型参数太少.



### (3)更新率

更新的权重占全部权值的比例.1e-3是一个不错的比例.低于这个值一般来说要调高学习率.

### (4)每层的激活值

画出柱状图,例如对于tanh,应该在[-1,1]之间.

### (5)可视化

可视化后的结果应该平整特征突出.

## 参数更新

### 普通更新

梯度影响位置.

x **+=** **-** learning\_rate **\*** dx

### 动量更新

梯度影响速度,速度影响位置.

v **=** mu **\*** v **-** learning\_rate **\*** dx *# integrate velocity*

x **+=** v *# integrate position*

### Nesterov动量

目前比普通梯度下降更好一些的方法.

## 退火学习率下降

学习率下降得太慢会徘徊不前,太快会无法到达最优解.3种方法:

1. step decay:每5步衰减0.5.或者观察val,每当val的error不再变化,learning rate就折半.
2. 指数衰减
3. 1/t衰减

实际主要用step decay.

## 逐参数适应学习率

### 1.Adagrad

cache **+=** dx**\*\***2

x **+=** **-** learning\_rate **\*** dx **/** (np**.**sqrt(cache) **+** eps)

获得高梯度的更新被减弱,低梯度增强,eps防止除以0,不能去掉sqrt否则表达能力减弱.

Downside:learning rate会单调下降.

### 2.RMSprop

cache **=** decay\_rate **\*** cache **+** (1 **-** decay\_rate) **\*** dx**\*\***2

x **+=** **-** learning\_rate **\*** dx **/** (np**.**sqrt(cache) **+** eps)

改进了cache部分,使得learning rate不再单调下降.

### 3.Adam

m **=** beta1**\***m **+** (1**-**beta1)**\***dx

v **=** beta2**\***v **+** (1**-**beta2)**\***(dx**\*\***2)

x **+=** **-** learning\_rate **\*** m **/** (np**.**sqrt(v) **+** eps)

比2,用了平滑版梯度.

## 超参数调优

初始化学习率

学习率衰减方式

正则项

### 1.实现问题

worker用来记录checkpoint(validation的表达准确率),一个master用来调控workers.

### 2.用一个验证集而不是交叉验证

### 3.超参数范围

对数范围取值.learning rate和regularization都是.因为他们都有乘的作用.

learning\_rate = 10 \*\* uniform(-6, 1)

dropout除外.

dropout = uniform(0,1)

### 4.随机搜索比网格搜索结果更好

经验之谈.

### 边界最优值

万一最优参数出现在边界上,要小心是不是错过了其他更好的参数,这一般发生在初始范围设定的不太好的前提之下.

### 从粗糙到细致地搜索

先在大范围搜索一个epoch,然后再细分.

## 模型集成

提升神经网络表达效果的一个可靠方式就是计算好几个独立模型的结果,然后求均值,模型数量越多,准确度越高,当然准确度提升的越慢.

1. 用最好的一组参数训练不同初始化权重的模型.
2. 用最好的几组参数训练几个模型.交叉验证之后就不需要再训练模型了.

3.就用训练超参数过程中的几个checkpoint对应的模型直接集成.很省事.

4.一旦这一次损失值相对于上一次出现指数下降,就记录下来权重,对这几个模型集成.效果总是很好.

模型集成的劣势是消耗时间,参考hinton的论文,从一个好的集成中抽一个单独的模型出来.

## 总结

参数的时候,用SGD+N动量法 或 Adam!

## Unknown word

Demystify 揭秘

Sanity 明智的

Stall 失速

Diagnose 诊断

Misnomer 用词不当

Friction 摩擦力

Damp 抑制

kinetic energy 动能

Annealing 退火

nudge 轻推

Lookahead 前瞻

Overshooting 超调

Miscellaneous 混杂的

Inspect 检查

Uniform distribution 标准分布

Exploitation 开发

Distill 提取

# Putting it together: Minimal Neural Network Case Study

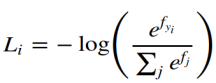
主要总结写2层NN的流程,主要包括:

* 初始化参数
* forward计算score(sigmoid/ReLU/tanh)
* 通过score和label计算loss(用SVM/softmax)
* 根据loss选择的公式,backward计算梯度

Ps:sigmoid函数如下,范围:[-1,1]:

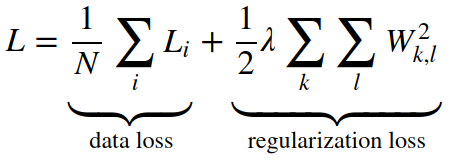
IMG_256

Softmax函数如下,范围:[0,1]:



## 计算loss

Loss包含data loss和regularization loss两个部分:



Tricks:计算正则项损失时,用0.5,使得bp过程更简单.

## 计算gradient

## Unknown word

# Assignment1\_Q4: Two-Layer Neural Network

## 一些问题

Jupiter报错

ModuleNotFoundError: No module named 'past'

解决:

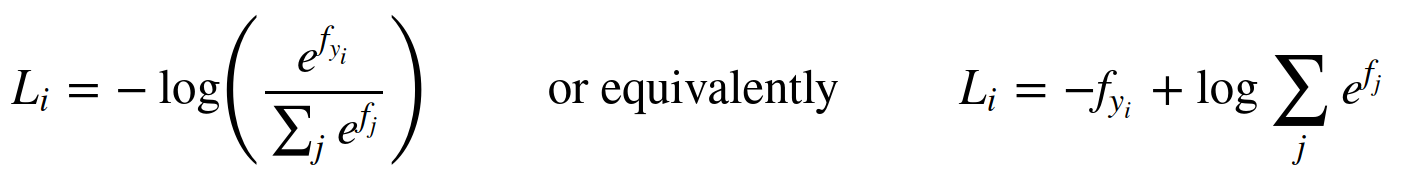
|  |
| --- |
| pip install future |

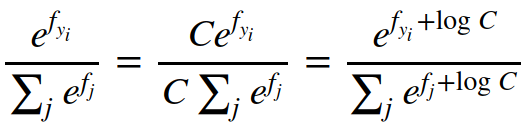
## 2.我的方法(my\_two\_layer\_net.ipynb和my\_neural\_net.py)

### 2.1 forward问题

第二层不需要做激励函数

Softmax Loss公式:



 2017-12-15 14-59-23屏幕截图

记得加正则项,正则项应该包含所有的W(W1和W2).

X:(N,D)

Y:(N,1)

W1: (D, H)

b1: (H,1)

W2: (H, C)

b2: (C,1)

### 2.2 Backward

|  |
| --- |
| # layer2  dh2 = exp\_class\_score / np.sum(exp\_class\_score, axis=1,keepdims=True)  dh2[np.arange(N),y]-=1  dh2/=N  dW2=np.dot(h1.T,dh2)  dW2+=2\*reg\*W2  db2=np.sum(dh2,axis=0)  #layer1  dh1=np.dot(dh2,W2.T)  dW1X\_b1=dh1  dW1X\_b1[h1<=0]=0  dW1=np.dot(X.T,dW1X\_b1)  dW1 += 2\*reg \* W1  db1=np.sum(dW1X\_b1,axis=0)  grads['W2']=dW2  grads['b2']=db2  grads['W1']=dW1  grads['b1'] = db1 |

## 3.根据官方note完成

Refer to: Putting it together: Minimal Neural Network Case Study

### 3.1 forward

|  |
| --- |
| # get unnormalized probabilities  exp\_scores = np.exp(scores)  # normalize them for each example  probs = exp\_scores / np.sum(exp\_scores, axis=1, keepdims=True)  corect\_logprobs = -np.log(probs[range(num\_examples), y])  data\_loss = np.sum(corect\_logprobs) / num\_examples  reg\_loss = 0.5 \* reg\*(np.sum(W2\*\*2)+np.sum(W1\*\*2)) |

### 3.2 backward

|  |
| --- |
| # layer2  dh2 = exp\_class\_score / np.sum(exp\_class\_score, axis=1,keepdims=True)  dh2[np.arange(N),y]-=1  dh2/=N  dW2=np.dot(h1.T,dh2)  dW2+=2\*reg\*W2  db2=np.sum(dh2,axis=0)  #layer1  dh1=np.dot(dh2,W2.T)  dW1X\_b1=dh1  dW1X\_b1[h1<=0]=0  dW1=np.dot(X.T,dW1X\_b1)  dW1 += 2\*reg \* W1  db1=np.sum(dW1X\_b1,axis=0)  grads['W2']=dW2  grads['b2']=db2  grads['W1']=dW1  grads['b1'] = db1 |

## Unknown word

Scalar 标量

# Assignment1\_Q5: Higher Level Representations: Image Features

## 1.核心Extract Features

(1)Histogram of Oriented Gradients (HOG)

无颜色纹理特征

(2)color histogram

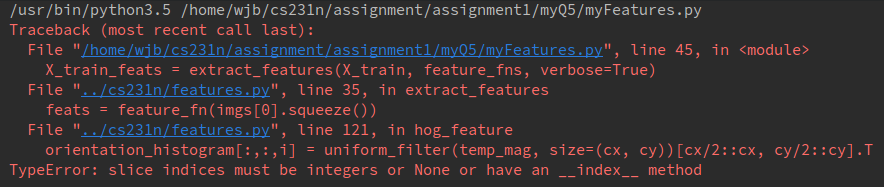
颜色特征

## Bug

运行这段代码:

|  |
| --- |
| from cs231n.features import \*  num\_color\_bins = 10 # Number of bins in the color histogram  feature\_fns = [hog\_feature, lambda img: color\_histogram\_hsv(img, nbin=num\_color\_bins)]  X\_train\_feats = extract\_features(X\_train, feature\_fns, verbose=True) |

报错:



TypeError: slice indices must be integers or None or have an index method

原因是在../cs231n/features.py里,121行这句:

|  |
| --- |
| orientation\_histogram[:,:,i] = uniform\_filter(temp\_mag, size=(cx, cy))[cx/2::cx, cy/2::cy].T |

里面cx/2会转换成浮点数,因此需要强制转换成int,也就是改成这样:

|  |
| --- |
| orientation\_histogram[:,:,i] = uniform\_filter(temp\_mag, size=(cx, cy))[int(cx/2)::cx, int(cy/2)::cy].T |

## Unknown word

Hue 色调

Ought 应该

Bonus 奖金

# 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?