每周学习进展阶段汇报

----CS231n Stanford University深度学习基础课程

汇报人 : 胡小婉

时间段 : 2018年2月26日(周一)至2018年3月4日(周日)

[1. image classification notes(图像分类) 2](#_Toc1344334379)

[1.1 图像分类、数据驱动方法和流程 2](#_Toc2125758104)

[1.2 Nearest Neighbor分类器 3](#_Toc768926596)

[1.3 k-Nearest Neighbor分类器 5](#_Toc876155249)

[1.4 验证集、交叉验证集和超参数调优 5](#_Toc1013557553)

[1.4.1 用于超参数调优的验证集 5](#_Toc857641044)

[1.4.2 交叉验证 6](#_Toc1651630842)

[1.4.3 实际应用 6](#_Toc1257666268)

[1.5 Nearest Neighbor分类器的优劣 7](#_Toc2019620804)

[1.6 小结 7](#_Toc1166642307)

[1.7 实际应用k-NN 7](#_Toc1313518965)

[2. linear classification notes(线性分类) 8](#_Toc863104032)

[2.1 线性分类器简介 8](#_Toc590340381)

[2.2 从图像到标签分值的参数化映射 8](#_Toc688894854)

[2.2.1 需要注意的几点 9](#_Toc1090277732)

[2.3 损失函数 Loss function 10](#_Toc706145831)

[2.3.1 多类支持向量机损失 Multiclass Support Vector Machine Loss 11](#_Toc1289303509)

[2.3.2 Softmax分类器 12](#_Toc1788554853)

[2.3.3 SVM和Softmax的比较 13](#_Toc1459609287)

[2.4 小结 14](#_Toc1880772336)

[3. optimization notes(最优化) 14](#_Toc925244103)

[3.1 简介 14](#_Toc2039194151)

[3.2 损失函数可视化 15](#_Toc1860903023)

[3.3 最优化 Optimization 15](#_Toc455730058)

[3.3.1 随机搜索 15](#_Toc1448651951)

[3.3.2 随机本地搜索 16](#_Toc1292481671)

[3.3.3 跟随梯度 16](#_Toc1253218859)

[3.3 梯度计算 17](#_Toc1117303167)

[Tips: 实践考量 17](#_Toc311746947)

[3.4 微分分析计算梯度 18](#_Toc159981665)

[3.5 梯度下降 19](#_Toc945107998)

[3.6 小结 19](#_Toc1656081326)

[4. backprop notes(反向传播算法) 20](#_Toc138256121)

[4.1 简介 20](#_Toc1714034595)

[4.2 使用链式法则计算复合表达式 20](#_Toc384752928)

[4.3 模块化：Sigmoid例子 21](#_Toc1151813674)

[4.4 反向传播实践：分段计算 22](#_Toc424191991)

[4.5 用向量化操作计算梯度 22](#_Toc2036383770)

[4.6 小结 23](#_Toc261996295)

[4. [Assignment #1]--作业一 23](#_Toc296329147)

[Q1: k-Nearest Neighbor classifier 23](#_Toc1055542429)

[Q2: Training a Support Vector Machine 26](#_Toc1575515260)

[Q3: Implement a Softmax classifier 31](#_Toc1159433179)

[Q4: Two-Layer Neural Network 35](#_Toc1645882811)

[Q5: Higher Level Representations: Image Features 41](#_Toc116926467)

[6. 对一些概念和函数的理解 45](#_Toc102227263)

[6.1 曼哈顿距离和欧氏距离比较 45](#_Toc204544994)

[6.2 numpy的ravel() 和 flatten()函数 45](#_Toc1406229976)

[6.3 np.linalg.norm(求范数) 46](#_Toc1890782116)

[6.4 Numpy.concatenate 46](#_Toc1664154281)

[6.5 numpy.flatnonzero() 47](#_Toc1139518664)

[6.6 np.random.seed(0) 47](#_Toc668542571)

[6.7 numpy的ravel() 和 flatten()函数 48](#_Toc1555864785)

[6.8 scipy.ndimage.filters.uniform\_filter(均值滤波) 48](#_Toc852938039)

[6.9 Numpy.linspace 48](#_Toc1124272630)

[6.10 np.where使用 49](#_Toc857033088)

[6.11 numpy diff函数 50](#_Toc2145419710)

[6.12 Np.histogram 50](#_Toc230007841)

# 1. image classification notes(图像分类)

## 1.1 图像分类、数据驱动方法和流程

目标：这一节我们将介绍图像分类问题。所谓图像分类问题，就是已有固定的分类标签集合，然后对于输入的图像，从分类标签集合中找出一个分类标签，最后把分类标签分配给该输入图像。虽然看起来挺简单的，但这可是计算机视觉领域的核心问题之一，并且有着各种各样的实际应用。在后面的课程中，我们可以看到计算机视觉领域中很多看似不同的问题（比如物体检测和分割），都可以被归结为图像分类问题。

图像分类的任务，就是对于一个给定的图像，预测它属于的那个分类标签（或者给出属于一系列不同标签的可能性）。图像是3维数组，数组元素是取值范围从0到255的整数。数组的尺寸是宽度x高度x3，其中这个3代表的是红、绿和蓝3个颜色通道。

困难和挑战：对于人来说，识别出一个像“猫”一样视觉概念是简单至极的，然而从计算机视觉算法的角度来看就值得深思了。我们在下面列举了计算机视觉算法在图像识别方面遇到的一些困难，要记住图像是以3维数组来表示的，数组中的元素是亮度值。

视角变化（Viewpoint variation）：同一个物体，摄像机可以从多个角度来展现。

大小变化（Scale variation）：物体可视的大小通常是会变化的（不仅是在图片中，在真实世界中大小也是变化的）。

形变（Deformation）：很多东西的形状并非一成不变，会有很大变化。

遮挡（Occlusion）：目标物体可能被挡住。有时候只有物体的一小部分（可以小到几个像素）是可见的。

光照条件（Illumination conditions）：在像素层面上，光照的影响非常大。

背景干扰（Background clutter）：物体可能混入背景之中，使之难以被辨认。

类内差异（Intra-class variation）：一类物体的个体之间的外形差异很大，比如椅子。这一类物体有许多不同的对象，每个都有自己的外形。

数据驱动方法：如何写一个图像分类的算法呢？这和写个排序算法可是大不一样。怎么写一个从图像中认出猫的算法？搞不清楚。因此，与其在代码中直接写明各类物体到底看起来是什么样的，倒不如说我们采取的方法和教小孩儿看图识物类似：给计算机很多数据，然后实现学习算法，让计算机学习到每个类的外形。这种方法，就是数据驱动方法



图像分类流程。在课程视频中已经学习过，图像分类就是输入一个元素为像素值的数组，然后给它分配一个分类标签。完整流程如下：

输入：输入是包含N个图像的集合，每个图像的标签是K种分类标签中的一种。这个集合称为训练集。

学习：这一步的任务是使用训练集来学习每个类到底长什么样。一般该步骤叫做训练分类器或者学习一个模型。

评价：让分类器来预测它未曾见过的图像的分类标签，并以此来评价分类器的质量。我们会把分类器预测的标签和图像真正的分类标签对比。毫无疑问，分类器预测的分类标签和图像真正的分类标签如果一致，那就是好事，这样的情况越多越好。

面对以上所有变化及其组合，好的图像分类模型能够在维持分类结论稳定的同时，保持对类间差异足够敏感。

## 1.2 Nearest Neighbor分类器

图像分类数据集：CIFAR-10



假设现在我们有CIFAR-10的50000张图片（每种分类5000张）作为训练集，我们希望将余下的10000作为测试集并给他们打上标签。Nearest Neighbor算法将会拿着测试图片和训练集中每一张图片去比较，然后将它认为最相似的那个训练集图片的标签赋给这张测试图片。

那么具体如何比较两张图片呢？在本例中，就是比较32x32x3的像素块。最简单的方法就是逐个像素比较，最后将差异值全部加起来。换句话说，就是将两张图片先转化为两个向量I1和I2，然后计算他们的L1距离：

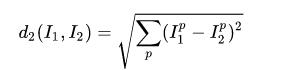


以图片中的一个颜色通道为例来进行说明。两张图片使用L1距离来进行比较。逐个像素求差值，然后将所有差值加起来得到一个数值。如果两张图片一模一样，那么L1距离为0，但是如果两张图片很是不同，那L1值将会非常大。



作为评价标准，我们常常使用准确率，它描述了我们预测正确的得分。请注意以后我们实现的所有分类器都需要有这个API：train(X, y)函数。该函数使用训练集的数据和标签来进行训练。从其内部来看，类应该实现一些关于标签和标签如何被预测的模型。这里还有个predict(X)函数，它的作用是预测输入的新数据的分类标签。

距离选择：计算向量间的距离有很多种方法，另一个常用的方法是L2距离，从几何学的角度，可以理解为它在计算两个向量间的欧式距离。L2距离的公式如下：



换句话说，我们依旧是在计算像素间的差值，只是先求其平方，然后把这些平方全部加起来，最后对这个和开方。在Numpy中，我们只需要替换上面代码中的1行代码就行：

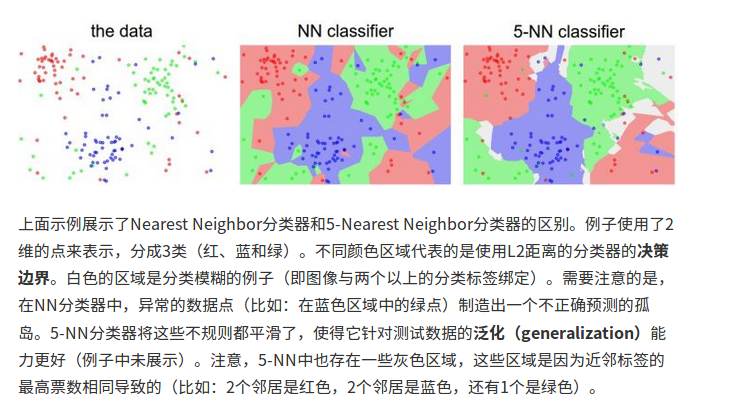
distances = np.sqrt(np.sum(np.square(self.Xtr - X[i,:]), axis = 1))

注意在这里使用了np.sqrt，但是在实际中可能不用。因为求平方根函数是一个单调函数，它对不同距离的绝对值求平方根虽然改变了数值大小，但依然保持了不同距离大小的顺序。所以用不用它，都能够对像素差异的大小进行正确比较。如果你在CIFAR-10上面跑这个模型，正确率是35.4%，比刚才低了一点。

L1和L2比较。比较这两个度量方式是挺有意思的。在面对两个向量之间的差异时，L2比L1更加不能容忍这些差异。也就是说，相对于1个巨大的差异，L2距离更倾向于接受多个中等程度的差异。L1和L2都是在p-norm常用的特殊形式。

## 1.3 k-Nearest Neighbor分类器

使用k-Nearest Neighbor分类器就能做得更好。它的思想很简单：与其只找最相近的那1个图片的标签，我们找最相似的k个图片的标签，然后让他们针对测试图片进行投票，最后把票数最高的标签作为对测试图片的预测。所以当k=1的时候，k-Nearest Neighbor分类器就是Nearest Neighbor分类器。从直观感受上就可以看到，更高的k值可以让分类的效果更平滑，使得分类器对于异常值更有抵抗力。



## 1.4 验证集、交叉验证集和超参数调优

### 1.4.1 用于超参数调优的验证集

k-NN分类器需要设定k值，那么选择哪个k值最合适的呢？我们可以选择不同的距离函数，比如L1范数和L2范数等，那么选哪个好？还有不少选择我们甚至连考虑都没有考虑到（比如：点积）。所有这些选择，被称为超参数（hyperparameter）。在基于数据进行学习的机器学习算法设计中，超参数是很常见的。一般说来，这些超参数具体怎么设置或取值并不是显而易见的。

你可能会建议尝试不同的值，看哪个值表现最好就选哪个。好主意！我们就是这么做的，但这样做的时候要非常细心。特别注意：决不能使用测试集来进行调优。当你在设计机器学习算法的时候，应该把测试集看做非常珍贵的资源，不到最后一步，绝不使用它。如果你使用测试集来调优，而且算法看起来效果不错，那么真正的危险在于：算法实际部署后，性能可能会远低于预期。这种情况，称之为算法对测试集过拟合。从另一个角度来说，如果使用测试集来调优，实际上就是把测试集当做训练集，由测试集训练出来的算法再跑测试集，自然性能看起来会很好。这其实是过于乐观了，实际部署起来效果就会差很多。所以，最终测试的时候再使用测试集，可以很好地近似度量你所设计的分类器的泛化性能（在接下来的课程中会有很多关于泛化性能的讨论）。

**测试数据集只使用一次，即在训练完成后评价最终的模型时使用。**

好在我们有不用测试集调优的方法。其思路是：从训练集中取出一部分数据用来调优，我们称之为验证集（validation set）。以CIFAR-10为例，我们可以用49000个图像作为训练集，用1000个图像作为验证集。验证集其实就是作为假的测试集来调优。

程序结束后，我们会作图分析出哪个k值表现最好，然后用这个k值来跑真正的测试集，并作出对算法的评价。

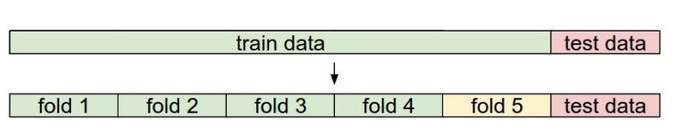
**把训练集分成训练集和验证集。使用验证集来对所有超参数调优。最后只在测试集上跑一次并报告结果**。

### 1.4.2 交叉验证

有时候，训练集数量较小（因此验证集的数量更小），人们会使用一种被称为交叉验证的方法，这种方法更加复杂些。还是用刚才的例子，如果是交叉验证集，我们就不是取1000个图像，而是将训练集平均分成5份，其中4份用来训练，1份用来验证。然后我们循环着取其中4份来训练，其中1份来验证，最后取所有5次验证结果的平均值作为算法验证结果。

### 1.4.3 实际应用

在实际情况下，人们不是很喜欢用交叉验证，主要是因为它会耗费较多的计算资源。一般直接把训练集按照50%-90%的比例分成训练集和验证集。但这也是根据具体情况来定的：如果超参数数量多，你可能就想用更大的验证集，而验证集的数量不够，那么最好还是用交叉验证吧。至于分成几份比较好，一般都是分成3、5和10份。



常用的数据分割模式。给出训练集和测试集后，训练集一般会被均分。这里是分成5份。前面4份用来训练，黄色那份用作验证集调优。如果采取交叉验证，那就各份轮流作为验证集。最后模型训练完毕，超参数都定好了，让模型跑一次（而且只跑一次）测试集，以此测试结果评价算法。

## 1.5 Nearest Neighbor分类器的优劣

现在对Nearest Neighbor分类器的优缺点进行思考。首先，Nearest Neighbor分类器易于理解，实现简单。其次，算法的训练不需要花时间，因为其训练过程只是将训练集数据存储起来。然而测试要花费大量时间计算，因为每个测试图像需要和所有存储的训练图像进行比较，这显然是一个缺点。在实际应用中，我们关注测试效率远远高于训练效率。其实，我们后续要学习的卷积神经网络在这个权衡上走到了另一个极端：虽然训练花费很多时间，但是一旦训练完成，对新的测试数据进行分类非常快。这样的模式就符合实际使用需求。

Nearest Neighbor分类器在某些特定情况（比如数据维度较低）下，可能是不错的选择。但是在实际的图像分类工作中，很少使用。因为图像都是高维度数据（他们通常包含很多像素），而高维度向量之间的距离通常是反直觉的。

## 1.6 小结

介绍了图像分类问题。在该问题中，给出一个由被标注了分类标签的图像组成的集合，要求算法能预测没有标签的图像的分类标签，并根据算法预测准确率进行评价。

介绍了一个简单的图像分类器：最近邻分类器(Nearest Neighbor classifier)。分类器中存在不同的超参数(比如k值或距离类型的选取)，要想选取好的超参数不是一件轻而易举的事。

选取超参数的正确方法是：将原始训练集分为训练集和验证集，我们在验证集上尝试不同的超参数，最后保留表现最好那个。

如果训练数据量不够，使用交叉验证方法，它能帮助我们在选取最优超参数的时候减少噪音。

一旦找到最优的超参数，就让算法以该参数在测试集跑且只跑一次，并根据测试结果评价算法。

最近邻分类器能够在CIFAR-10上得到将近40%的准确率。该算法简单易实现，但需要存储所有训练数据，并且在测试的时候过于耗费计算能力。

最后，我们知道了仅仅使用L1和L2范数来进行像素比较是不够的，图像更多的是按照背景和颜色被分类，而不是语义主体分身。

## 1.7 实际应用k-NN

按照以下流程：

1.预处理你的数据：对你数据中的特征进行归一化（normalize），让其具有零平均值（zero mean）和单位方差（unit variance）。

2. 如果数据是高维数据，考虑使用降维方法，比如PCA(wiki ref, CS229ref, blog ref)或随机投影。

3. 将数据随机分入训练集和验证集。按照一般规律，70%-90% 数据作为训练集。这个比例根据算法中有多少超参数，以及这些超参数对于算法的预期影响来决定。如果需要预测的超参数很多，那么就应该使用更大的验证集来有效地估计它们。如果担心验证集数量不够，那么就尝试交叉验证方法。如果计算资源足够，使用交叉验证总是更加安全的（份数越多，效果越好，也更耗费计算资源）。

4. 在验证集上调优，尝试足够多的k值，尝试L1和L2两种范数计算方式。

5.如果分类器跑得太慢，尝试使用Approximate Nearest Neighbor库（比如FLANN）来加速这个过程，其代价是降低一些准确率。

6..对最优的超参数做记录。记录最优参数后，是否应该让使用最优参数的算法在完整的训练集上运行并再次训练呢？因为如果把验证集重新放回到训练集中（自然训练集的数据量就又变大了），有可能最优参数又会有所变化。在实践中，不要这样做。千万不要在最终的分类器中使用验证集数据，这样做会破坏对于最优参数的估计。直接使用测试集来测试用最优参数设置好的最优模型，得到测试集数据的分类准确率，并以此作为你的kNN分类器在该数据上的性能表现。

# 2. linear classification notes(线性分类)

## 2.1 线性分类器简介

图像分类的任务，就是从已有的固定分类标签集合中选择一个并分配给一张图像。我们还介绍了k-Nearest Neighbor （k-NN）分类器，该分类器的基本思想是通过将测试图像与训练集带标签的图像进行比较，来给测试图像打上分类标签。k-Nearest Neighbor分类器存在以下不足：

分类器必须记住所有训练数据并将其存储起来，以便于未来测试数据用于比较。这在存储空间上是低效的，数据集的大小很容易就以GB计。

对一个测试图像进行分类需要和所有训练图像作比较，算法计算资源耗费高。

概述：我们将要实现一种更强大的方法来解决图像分类问题，该方法可以自然地延伸到神经网络和卷积神经网络上。这种方法主要有两部分组成：一个是**评分函数（score function）**，它是原始图像数据到类别分值的映射。另一个是**损失函数（loss function）**，它是用来量化预测分类标签的得分与真实标签之间一致性的。该方法可转化为一个最优化问题，在最优化过程中，将通过更新评分函数的参数来最小化损失函数值。

## 2.2 从图像到标签分值的参数化映射

该方法的第一部分就是定义一个评分函数，这个函数将图像的像素值映射为各个分类类别的得分，得分高低代表图像属于该类别的可能性高低。下面会利用一个具体例子来展示该方法。现在假设有一个包含很多图像的训练集每个图像都有一个对应的分类标签y\_i。这里i=1,2...N并且。这就是说，我们有N个图像样例，每个图像的维度是D，共有K种不同的分类。

举例来说，在CIFAR-10中，我们有一个N=50000的训练集，每个图像有D=32x32x3=3072个像素，而K=10，这是因为图片被分为10个不同的类别（狗，猫，汽车等）。我们现在定义评分函数为：，该函数是原始图像像素到分类分值的映射。

线性分类器：在本模型中，我们从最简单的概率函数开始，一个线性映射：



在上面的公式中，假设每个图像数据都被拉长为一个长度为D的列向量，大小为[D x 1]。其中大小为[K x D]的矩阵W和大小为[K x 1]列向量b为该函数的参数（parameters）。还是以CIFAR-10为例，x\_i就包含了第i个图像的所有像素信息，这些信息被拉成为一个[3072 x 1]的列向量，W大小为[10x3072]，b的大小为[10x1]。因此，3072个数字（原始像素数值）输入函数，函数输出10个数字（不同分类得到的分值）。参数W被称为**权重（weights）**。b被称为**偏差向量（bias vector）**，这是因为它影响输出数值，但是并不和原始数据x\_i产生关联。在实际情况中，人们常常混用权重和参数这两个术语。

### 2.2.1 需要注意的几点

首先，一个单独的矩阵乘法Wx\_i就高效地并行评估10个不同的分类器（每个分类器针对一个分类），其中每个类的分类器就是W的一个行向量。

注意我们认为输入数据(x\_i,y\_i)是给定且不可改变的，但参数W和b是可控制改变的。我们的目标就是通过设置这些参数，使得计算出来的分类分值情况和训练集中图像数据的真实类别标签相符。在接下来的课程中，我们将详细介绍如何做到这一点，但是目前只需要直观地让正确分类的分值比错误分类的分值高即可。

该方法的一个优势是训练数据是用来学习到参数W和b的，一旦训练完成，训练数据就可以丢弃，留下学习到的参数即可。这是因为一个测试图像可以简单地输入函数，并基于计算出的分类分值来进行分类。

最后，注意只需要做一个矩阵乘法和一个矩阵加法就能对一个测试数据分类，这比k-NN中将测试图像和所有训练数据做比较的方法快多了。

**将图像看做高维度的点：**既然图像被伸展成为了一个高维度的列向量，那么我们可以把图像看做这个高维度空间中的一个点（即每张图像是3072维空间中的一个点）。整个数据集就是一个点的集合，每个点都带有1个分类标签。

既然定义每个分类类别的分值是权重和图像的矩阵乘，那么每个分类类别的分数就是这个空间中的一个线性函数的函数值。我们没办法可视化3072维空间中的线性函数，但假设把这些维度挤压到二维，那么就可以看看这些分类器在做什么了：



图像空间的示意图。其中每个图像是一个点，有3个分类器。以红色的汽车分类器为例，红线表示空间中汽车分类分数为0的点的集合，红色的箭头表示分值上升的方向。所有红线右边的点的分数值均为正，且线性升高。红线左边的点分值为负，且线性降低。

**将线性分类器看做模板匹配：**关于权重W的另一个解释是它的每一行对应着一个分类的模板（有时候也叫作原型）。一张图像对应不同分类的得分，是通过使用内积（也叫点积）来比较图像和模板，然后找到和哪个模板最相似。从这个角度来看，线性分类器就是在利用学习到的模板，针对图像做模板匹配。从另一个角度来看，可以认为还是在高效地使用k-NN，不同的是我们没有使用所有的训练集的图像来比较，而是每个类别只用了一张图片（这张图片是我们学习到的，而不是训练集中的某一张），而且我们会使用（负）内积来计算向量间的距离，而不是使用L1或者L2距离。

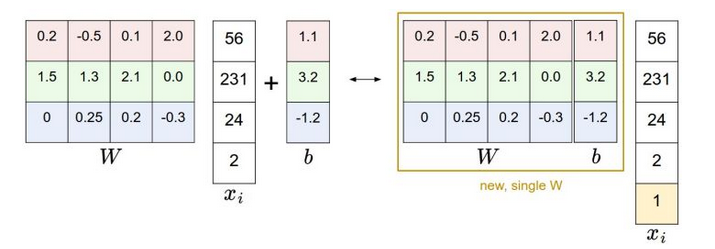
**偏差和权重的合并技巧：**在进一步学习前，要提一下这个经常使用的技巧。它能够将我们常用的参数W和b合二为一。回忆一下，分类评分函数定义为：



分开处理这两个参数（权重参数W和偏差参数b）有点笨拙，一般常用的方法是把两个参数放到同一个矩阵中，同时x\_i向量就要增加一个维度，这个维度的数值是常量1，这就是默认的偏差维度。这样新的公式就简化成下面这样：



还是以CIFAR-10为例，那么x\_i的大小就变成[3073x1]，而不是[3072x1]了，多出了包含常量1的1个维度）。W大小就是[10x3073]了。W中多出来的这一列对应的就是偏差值b，具体见下图：



偏差技巧的示意图。左边是先做矩阵乘法然后做加法，右边是将所有输入向量的维度增加1个含常量1的维度，并且在权重矩阵中增加一个偏差列，最后做一个矩阵乘法即可。左右是等价的。通过右边这样做，我们就只需要学习一个权重矩阵，而不用去学习两个分别装着权重和偏差的矩阵了。

**图像数据预处理：**在上面的例子中，所有图像都是使用的原始像素值（从0到255）。在机器学习中，对于输入的特征做归一化（normalization）处理是常见的套路。而在图像分类的例子中，图像上的每个像素可以看做一个特征。在实践中，对每个特征减去平均值来中心化数据是非常重要的。在这些图片的例子中，该步骤意味着根据训练集中所有的图像计算出一个平均图像值，然后每个图像都减去这个平均值，这样图像的像素值就大约分布在[-127, 127]之间了。下一个常见步骤是，让所有数值分布的区间变为[-1, 1]。零均值的中心化是很重要的

## 2.3 损失函数 Loss function

从图像像素值到所属类别的评分函数（score function），该函数的参数是权重矩阵W。在函数中，数据(x\_i,y\_i)是给定的，不能修改。但是我们可以调整权重矩阵这个参数，使得评分函数的结果与训练数据集中图像的真实类别一致，即评分函数在正确的分类的位置应当得到最高的评分（score）。

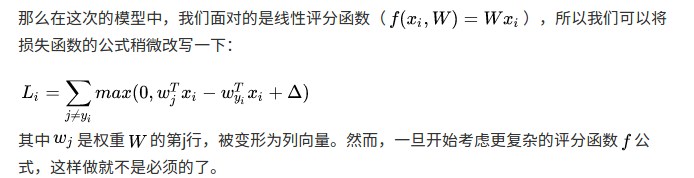
我们将使用损失函数（Loss Function）（有时也叫代价函数Cost Function或目标函数Objective）来衡量我们对结果的不满意程度。直观地讲，当评分函数输出结果与真实结果之间差异越大，损失函数输出越大，反之越小。

### 2.3.1 多类支持向量机损失 Multiclass Support Vector Machine Loss

损失函数的具体形式多种多样。首先，介绍常用的多类支持向量机（SVM）损失函数。SVM的损失函数想要SVM在正确分类上的得分始终比不正确分类上的得分高出一个边界值△,我们可以把损失函数想象成一个人，这位SVM先生（或者女士）对于结果有自己的品位，如果某个结果能使得损失值更低，那么SVM就更加喜欢它。

让我们更精确一些。回忆一下，第i个数据中包含图像x\_i的像素和代表正确类别的标签y\_i。评分函数输入像素数据，然后通过公式f(x\_i,W)来计算不同分类类别的分值。这里我们将分值简写为s。比如，针对第j个类别的得分就是第j个元素：针对第i个数据的多类SVM的损失函数定义如下：





关于0的阀值：max(0,-)函数，它常被称为折叶损失（hinge loss）。有时候会听到人们使用平方折叶损失SVM（即L2-SVM），它使用的是max(0,-)^2，将更强烈（平方地而不是线性地）地惩罚过界的边界值。不使用平方是更标准的版本，但是在某些数据集中，平方折叶损失会工作得更好。可以通过交叉验证来决定到底使用哪个。

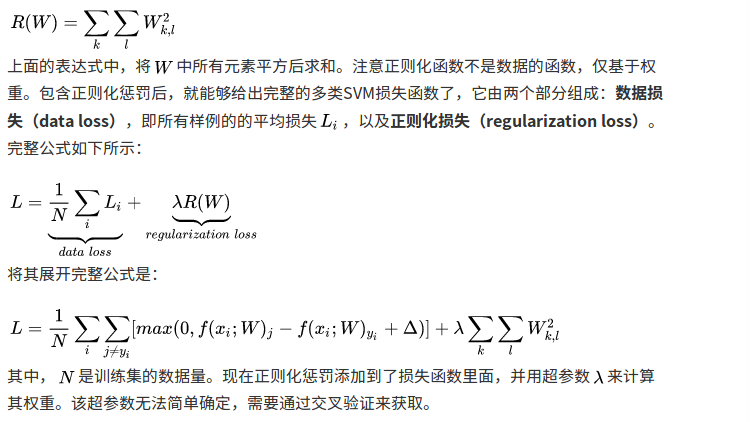
**我们对于预测训练集数据分类标签的情况总有一些不满意的，而损失函数就能将这些不满意的程度量化。**



多类SVM“想要”正确类别的分类分数比其他不正确分类类别的分数要高，而且至少高出delta的边界值。如果其他分类分数进入了红色的区域，甚至更高，那么就开始计算损失。如果没有这些情况，损失值为0。我们的目标是找到一些权重，它们既能够让训练集中的数据样例满足这些限制，也能让总的损失值尽可能地低.

**正则化（Regularization）：**上面损失函数有一个问题。假设有一个数据集和一个权重集W能够正确地分类每个数据（即所有的边界都满足，对于所有的i都有L\_i=0）。问题在于这个W并不唯一：可能有很多相似的W都能正确地分类所有的数据。一个简单的例子：如果W能够正确分类所有数据，即对于每个数据，损失值都是0。那么当λ>1时，任何数乘λW都能使得损失值为0，因为这个变化将所有分值的大小都均等地扩大了，所以它们之间的绝对差值也扩大了。举个例子，如果一个正确分类的分值和举例它最近的错误分类的分值的差距是15，对W乘以2将使得差距变成30。

换句话说，我们希望能向某些特定的权重W添加一些偏好，对其他权重则不添加，以此来消除模糊性。这一点是能够实现的，方法是向损失函数增加一个正**则化惩罚（regularization penalty）R(W)**部分。最常用的正则化惩罚是L2范式，L2范式通过对所有参数进行逐元素的平方惩罚来抑制大数值的权重：

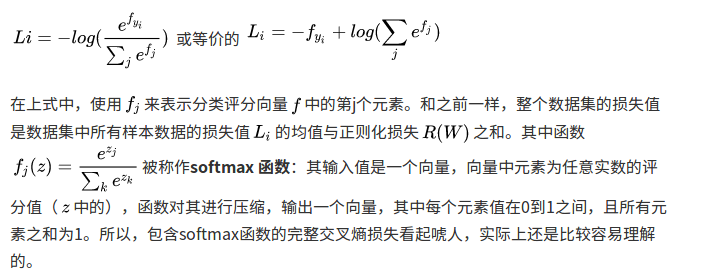


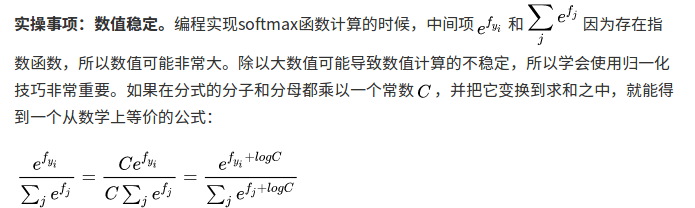
Tips: 实际考虑

设置Delta：你可能注意到上面的内容对超参数△及其设置是一笔带过，那么它应该被设置成什么值？需要通过交叉验证来求得吗？现在看来，该超参数在绝大多数情况下设为△=1.0都是安全的。超参数△和λ看起来是两个不同的超参数，但实际上他们一起控制同一个权衡：即损失函数中的数据损失和正则化损失之间的权衡。理解这一点的关键是要知道，权重W的大小对于分类分值有直接影响（当然对他们的差异也有直接影响）：当我们将W中值缩小，分类分值之间的差异也变小，反之亦然。因此，不同分类分值之间的边界的具体值（比如△=1或△=100）从某些角度来看是没意义的，因为权重自己就可以控制差异变大和缩小。也就是说，真正的权衡是我们允许权重能够变大到何种程度（通过正则化强λ来控制）。

### 2.3.2 Softmax分类器

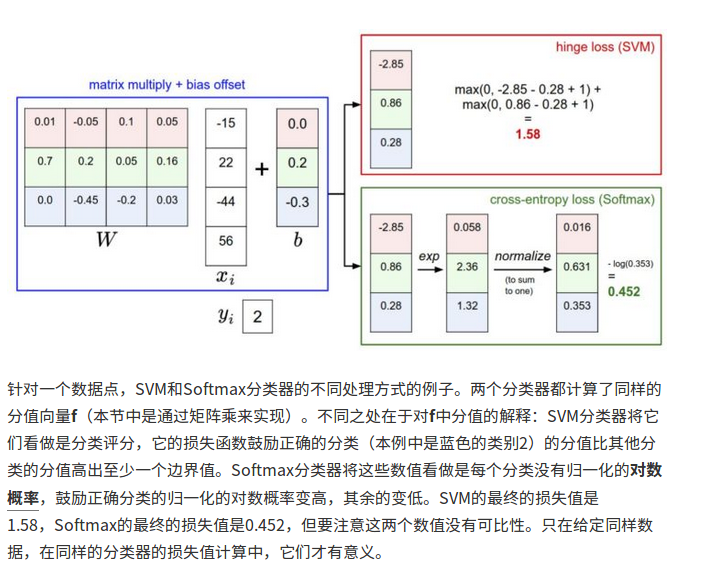
SVM是最常用的两个分类器之一，而另一个就是Softmax分类器，它的损失函数与SVM的损失函数不同。对于学习过二元逻辑回归分类器的读者来说，Softmax分类器就可以理解为逻辑回归分类器面对多个分类的一般化归纳。SVM将输出f(x\_i,W)作为每个分类的评分（因为无定标，所以难以直接解释）。与SVM不同，Softmax的输出（归一化的分类概率）更加直观，并且从概率上可以解释，这一点后文会讨论。在Softmax分类器中，函数映射f(x\_i;W)=Wx\_i保持不变，但将这些评分值视为每个分类的未归一化的对数概率，并且将折叶损失（hinge loss）替换为交叉熵损失（cross-entropy loss）。公式如下：





C的值可自由选择，不会影响计算结果，通过使用这个技巧可以提高计算中的数值稳定性。通常将C设为logC=-max\_jf\_j。该技巧简单地说，就是应该将向量f中的数值进行平移，使得最大值为0。

### 2.3.3 SVM和Softmax的比较



Softmax分类器为每个分类提供了“可能性”：SVM的计算是无标定的，而且难以针对所有分类的评分值给出直观解释。Softmax分类器则不同，它允许我们计算出对于所有分类标签的可能性。

## 2.4 小结

总结如下：

1. 定义了从图像像素映射到不同类别的分类评分的评分函数。在本节中，评分函数是一个基于权重W和偏差b的线性函数。

2. 与kNN分类器不同，参数方法的优势在于一旦通过训练学习到了参数，就可以将训练数据丢弃了。同时该方法对于新的测试数据的预测非常快，因为只需要与权重W进行一个矩阵乘法运算。

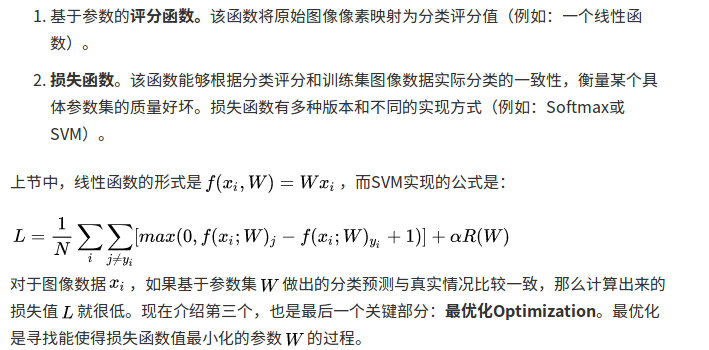
3. 介绍了偏差技巧，让我们能够将偏差向量和权重矩阵合二为一，然后就可以只跟踪一个矩阵。

4. 定义了损失函数（介绍了SVM和Softmax线性分类器最常用的2个损失函数）。损失函数能够衡量给出的参数集与训练集数据真实类别情况之间的一致性。在损失函数的定义中可以看到，对训练集数据做出良好预测与得到一个足够低的损失值这两件事是等价的。

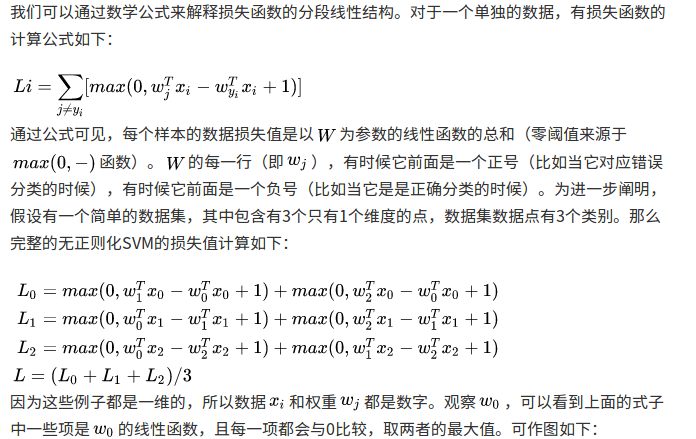
现在我们知道了如何基于参数，将数据集中的图像映射成为分类的评分，也知道了两种不同的损失函数，它们都能用来衡量算法分类预测的质量。但是，如何高效地得到能够使损失值最小的参数呢？这个求得最优参数的过程被称为最优化

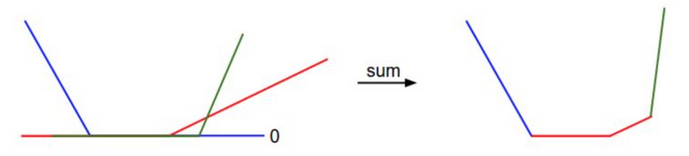
# 3. optimization notes(最优化)

## 3.1 简介



## 3.2 损失函数可视化





从一个维度方向上对数据损失值的展示。x轴方向就是一个权重，y轴就是损失值。数据损失是多个部分组合而成。其中每个部分要么是某个权重的独立部分，要么是该权重的线性函数与0阈值的比较。完整的SVM数据损失就是这个形状的30730维版本

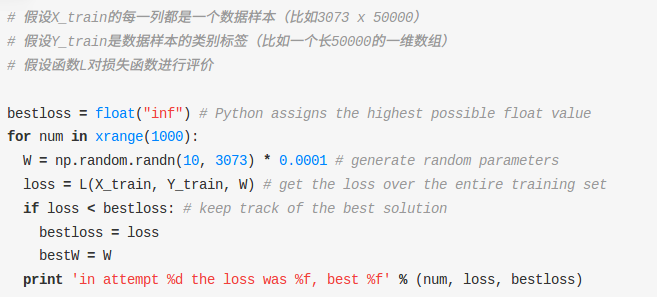
**不可导的损失函数:** 由于max操作，损失函数中存在一些不可导点（kinks），这些点使得损失函数不可微，因为在这些不可导点，梯度是没有定义的。但是次梯度（subgradient）依然存在且常常被使用

## 3.3 最优化 Optimization

损失函数可以量化某个具体权重集W的质量。而最优化的目标就是找到能够最小化损失函数值的W 。我们现在就朝着这个目标前进，实现一个能够最优化损失函数的方法。

### 3.3.1 随机搜索

既然确认参数集W的好坏蛮简单的，那第一个想到的方法，就是可以随机尝试很多不同的权重，然后看其中哪个最好。过程如下



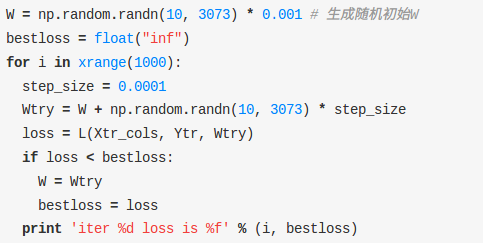
在上面的代码中，我们尝试了若干随机生成的权重矩阵W，其中某些的损失值较小，而另一些的损失值大些。我们可以把这次随机搜索中找到的最好的权重W取出，然后去跑测试集

**核心思路：迭代优化:**虽然找到最优的权重W非常困难，甚至是不可能的（尤其当W中存的是整个神经网络的权重的时候），但如果问题转化为：对一个权重矩阵集W取优，使其损失值稍微减少。那么问题的难度就大大降低了。换句话说，我们的方法从一个随机的W开始，然后对其迭代取优，每次都让它的损失值变得更小一点。

**从随机权重开始，然后迭代取优，从而获得更低的损失值。**

### 3.3.2 随机本地搜索

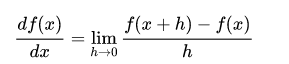
第一个策略可以看做是每走一步都尝试几个随机方向，如果某个方向是向山下的，就向该方向走一步。这次我们从一个随机W开始，然后生成一个随机的扰动\delta W ，只有当W+\delta W的损失值变低，我们才会更新。这个过程的具体代码如下：



### 3.3.3 跟随梯度

前两个策略中，我们是尝试在权重空间中找到一个方向，沿着该方向能降低损失函数的损失值。其实不需要随机寻找方向，因为可以直接计算出最好的方向，这就是从数学上计算出最陡峭的方向。这个方向就是损失函数的梯度（gradient）。在蒙眼徒步者的比喻中，这个方法就好比是感受我们脚下山体的倾斜程度，然后向着最陡峭的下降方向下山。

在一维函数中，斜率是函数在某一点的瞬时变化率。梯度是函数的斜率的一般化表达，它不是一个值，而是一个向量。在输入空间中，梯度是各个维度的斜率组成的向量（或者称为导数**derivatives**）。对一维函数的求导公式如下：



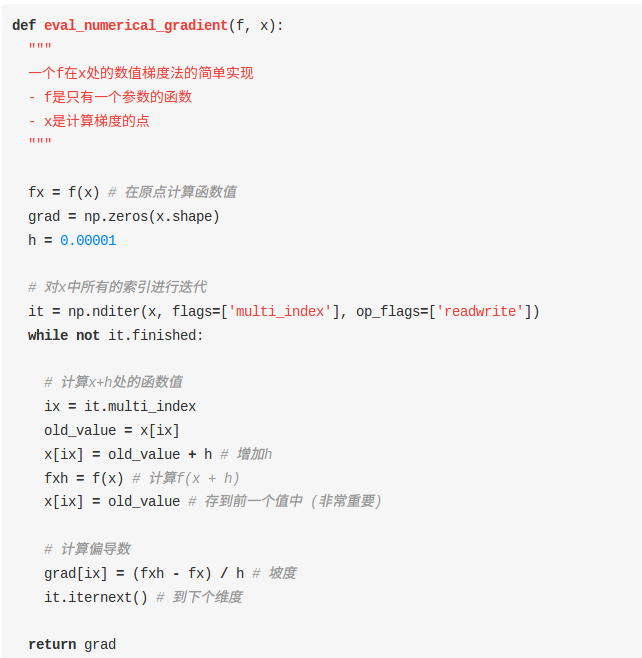
当函数有多个参数的时候，我们称导数为偏导数。而梯度就是在每个维度上偏导数所形成的向量。

## 3.3 梯度计算

计算梯度有两种方法：一个是缓慢的近似方法（**数值梯度法**），但实现相对简单。另一个方法（**分析梯度法**）计算迅速，结果精确，但是实现时容易出错，且需要使用微分。现在对两种方法进行介绍：

利用有限差值计算梯度

上节中的公式已经给出数值计算梯度的方法。下面代码是一个输入为函数f和向量x，计算f的梯度的通用函数，它返回函数f在点x处的梯度：



Tips: 实践考量

注意在数学公式中，h的取值是趋近于0的，然而在实际中，用一个很小的数值（比如例子中的1e-5）就足够了。在不产生数值计算出错的理想前提下，你会使用尽可能小的h。还有，实际中用中心差值公式效果较好

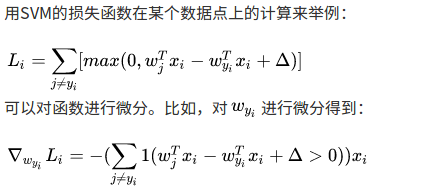


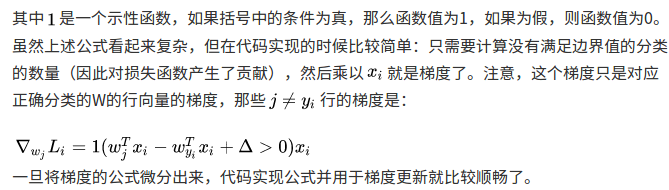
**在梯度负方向上更新：**在上面的代码中，为了计算W\_new，要注意我们是向着梯度df的负方向去更新，这是因为我们希望损失函数值是降低而不是升高。

**步长**的影响：梯度指明了函数在哪个方向是变化率最大的，但是没有指明在这个方向上应该走多远。选择步长（也叫作**学习率**）将会是神经网络训练中最重要（也是最头痛）的超参数设定之一。还是用蒙眼徒步者下山的比喻，这就好比我们可以感觉到脚朝向的不同方向上，地形的倾斜程度不同。但是该跨出多长的步长呢？不确定。如果谨慎地小步走，情况可能比较稳定但是进展较慢（这就是步长较小的情况）。相反，如果想尽快下山，那就大步走吧，但结果也不一定尽如人意。在上面的代码中就能看见反例，在某些点如果步长过大，反而可能越过最低点导致更高的损失值。

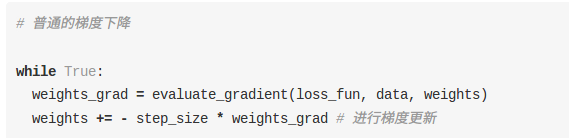
## 3.4 微分分析计算梯度

使用有限差值近似计算梯度比较简单，但缺点在于终究只是近似（因为我们对于h值是选取了一个很小的数值，但真正的梯度定义中h趋向0的极限），且耗费计算资源太多。第二个梯度计算方法是利用微分来分析，能得到计算梯度的公式（不是近似），用公式计算梯度速度很快，唯一不好的就是实现的时候容易出错。为了解决这个问题，在实际操作时常常将分析梯度法的结果和数值梯度法的结果作比较，以此来检查其实现的正确性，这个步骤叫做**梯度检查**。





## 3.5 梯度下降

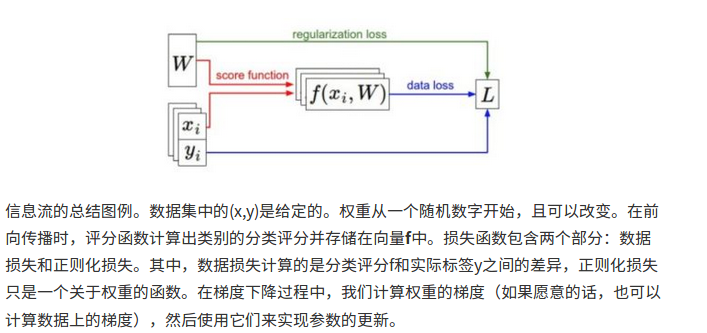


**小批量数据梯度下降（Mini-batch gradient descent）：**在大规模的应用中,训练数据可以达到百万级量级。如果像这样计算整个训练集，来获得仅仅一个参数的更新就太浪费了。一个常用的方法是计算训练集中的小批量（batches）数据。例如，在目前最高水平的卷积神经网络中，一个典型的小批量包含256个例子，而整个训练集是多少呢？一百二十万个。这个小批量数据就用来实现一个参数更新：



小批量数据策略有个极端情况，那就是每个批量中只有1个数据样本，这种策略被称为随机梯度下降（Stochastic Gradient Descent 简称**SGD**），有时候也被称为在线梯度下降

## 3.6 小结



1. 将损失函数比作了一个高维度的最优化地形，并尝试到达它的最底部。最优化的工作过程可以看做一个蒙着眼睛的徒步者希望摸索着走到山的底部。在例子中，可见SVM的损失函数是分段线性的，并且是碗状的。

2. 提出了迭代优化的思想，从一个随机的权重开始，然后一步步地让损失值变小，直到最小。

3. 函数的梯度给出了该函数最陡峭的上升方向。介绍了利用有限的差值来近似计算梯度的方法，该方法实现简单但是效率较低（有限差值就是h，用来计算数值梯度）。

4. 参数更新需要有技巧地设置步长。也叫学习率。如果步长太小，进度稳定但是缓慢，如果步长太大，进度快但是可能有风险。

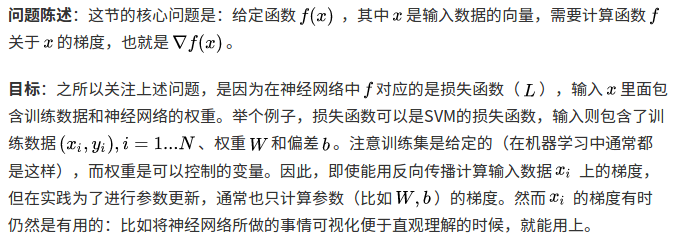
5.讨论权衡了数值梯度法和分析梯度法。数值梯度法计算简单，但结果只是近似且耗费计算资源。分析梯度法计算准确迅速但是实现容易出错，而且需要对梯度公式进行推导的数学基本功。因此，在实际中使用分析梯度法，然后使用梯度检查来检查其实现正确与否，其本质就是将分析梯度法的结果与数值梯度法的计算结果对比。

1. 介绍了梯度下降算法，它在循环中迭代地计算梯度并更新参数。

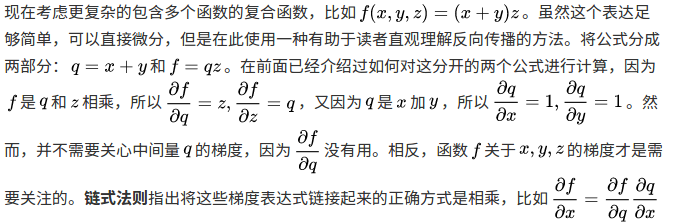
# 4. backprop notes(反向传播算法)

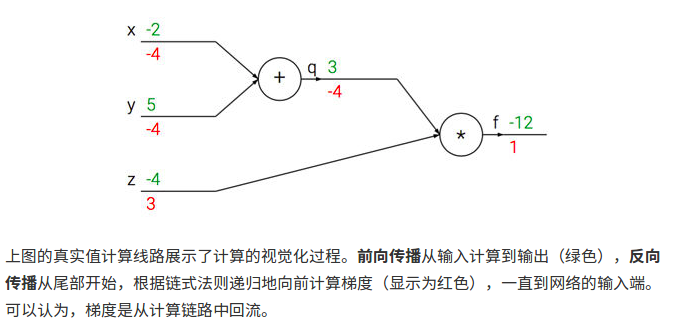
## 4.1 简介

反向传播是利用链式法则递归计算表达式的梯度的方法。理解反向传播过程及其精妙之处，对于理解、实现、设计和调试神经网络非常关键。



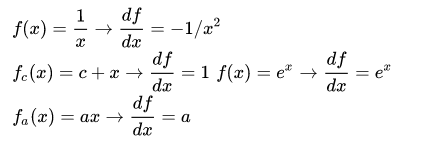
## 4.2 使用链式法则计算复合表达式

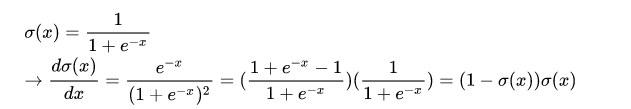


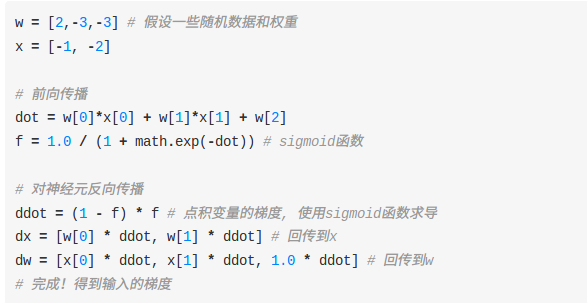


## 4.3 模块化：Sigmoid例子

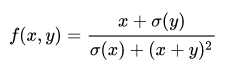








## 4.4 反向传播实践：分段计算



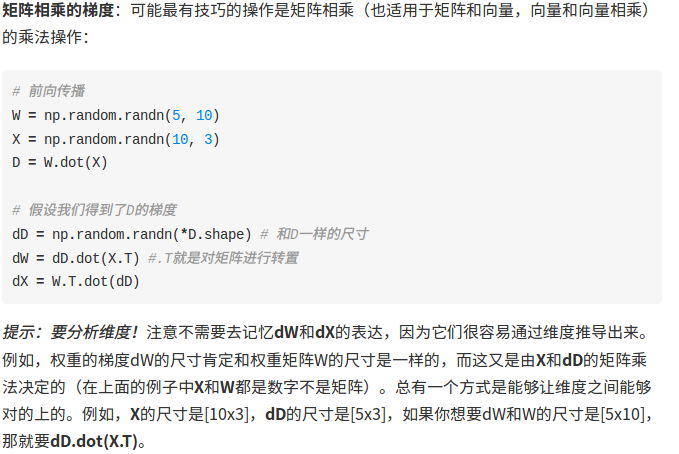
需要注意的一些东西：

**对前向传播变量进行缓存**：在计算反向传播时，前向传播过程中得到的一些中间变量非常有用。在实际操作中，最好代码实现对于这些中间变量的缓存，这样在反向传播的时候也能用上它们。如果这样做过于困难，也可以（但是浪费计算资源）重新计算它们。

**在不同分支的梯度要相加**：如果变量x，y在前向传播的表达式中出现多次，那么进行反向传播的时候就要非常小心，使用+=而不是=来累计这些变量的梯度（不然就会造成覆写）。这是遵循了在微积分中的多元链式法则，该法则指出如果变量在线路中分支走向不同的部分，那么梯度在回传的时候，就应该进行累加。

## 4.5 用向量化操作计算梯度

所有概念都适用于矩阵和向量操作。然而，在操作的时候要注意关注维度和转置操作。



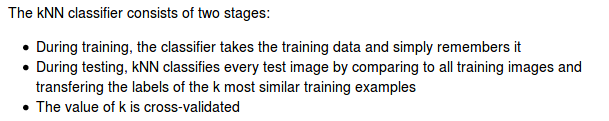
## 4.6 小结

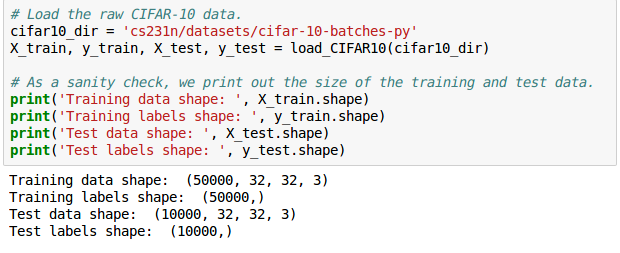
1.对梯度的含义有了直观理解，知道了梯度是如何在网络中反向传播的，知道了它们是如何与网络的不同部分通信并控制其升高或者降低，并使得最终输出值更高的。

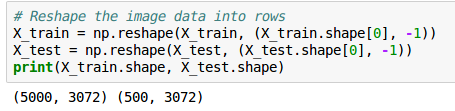
2. 讨论了分段计算在反向传播的实现中的重要性。应该将函数分成不同的模块，这样计算局部梯度相对容易，然后基于链式法则将其“链”起来。重要的是，不需要把这些表达式写在纸上然后演算它的完整求导公式，因为实际上并不需要关于输入变量的梯度的数学公式。只需要将表达式分成不同的可以求导的模块（模块可以是矩阵向量的乘法操作，或者取最大值操作，或者加法操作等），然后在反向传播中一步一步地计算梯度。

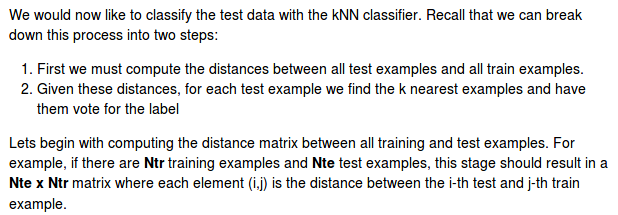
# 4. [Assignment #1]--作业一

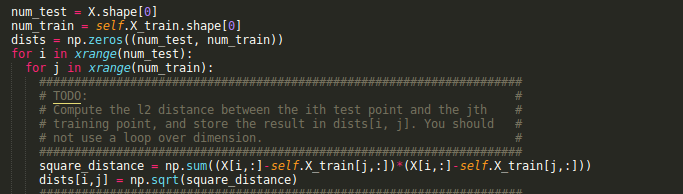
## Q1: k-Nearest Neighbor classifier

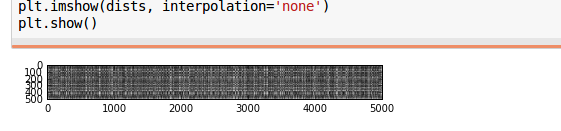




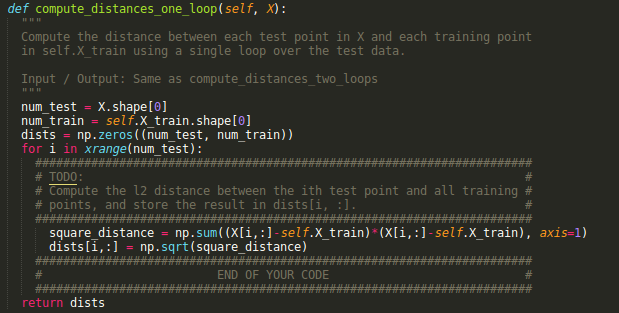


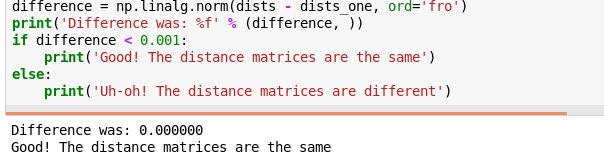


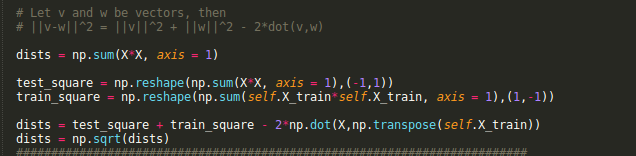




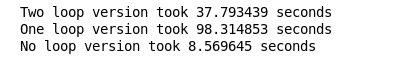




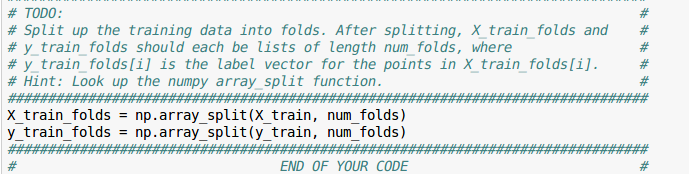




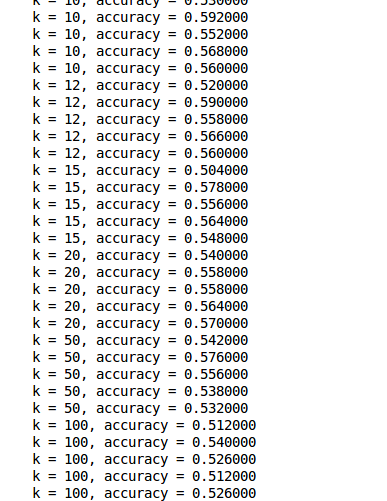
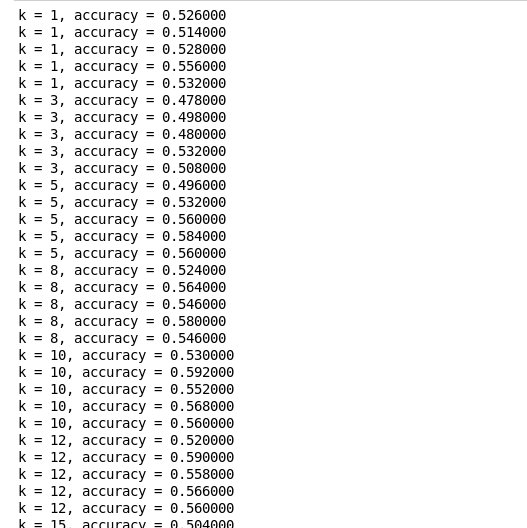
时间对比:

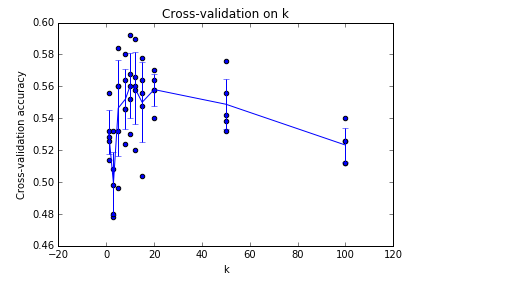


**Cross-validation :**



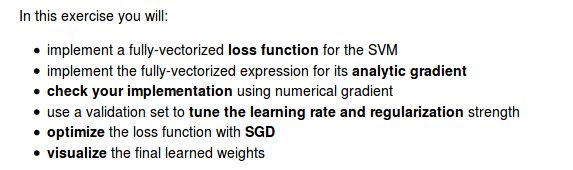


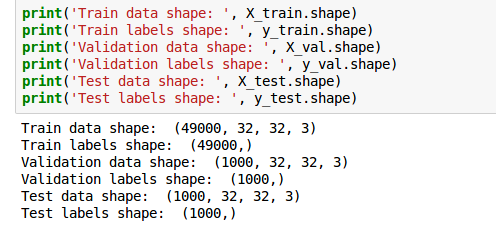


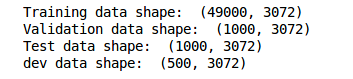




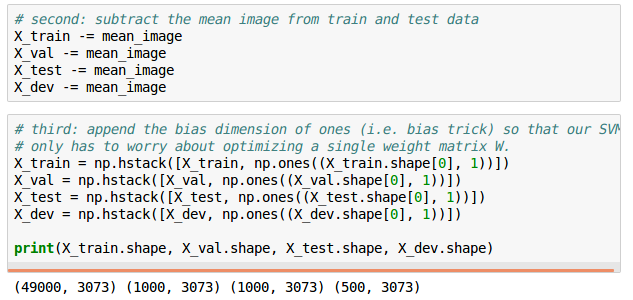
## Q2: Training a Support Vector Machine







预处理:

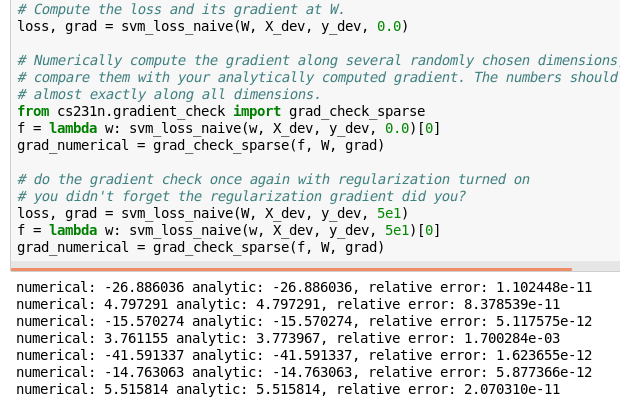


**SVM Classifier:**

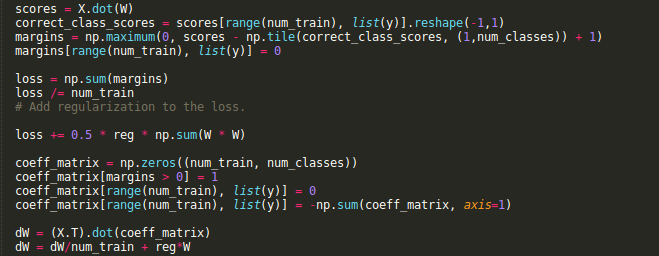




**梯度检查:**

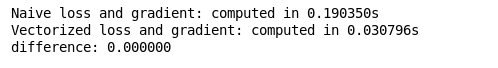


向量实现:

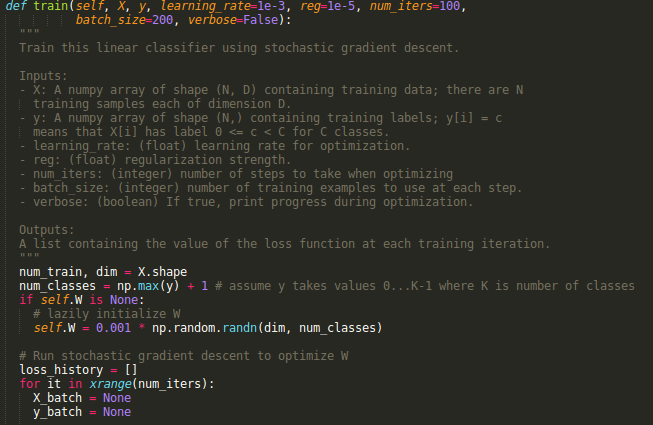


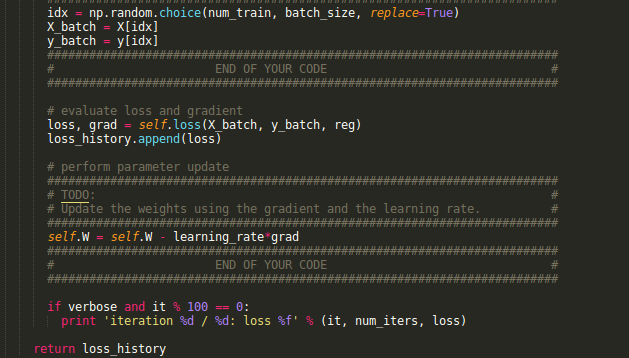
结果对比:

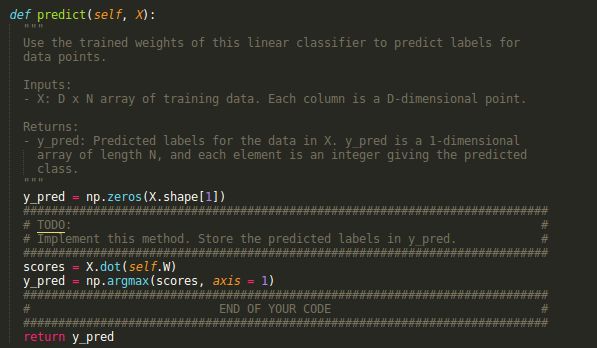


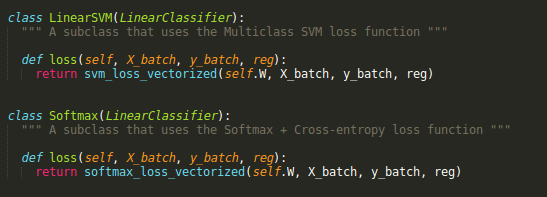


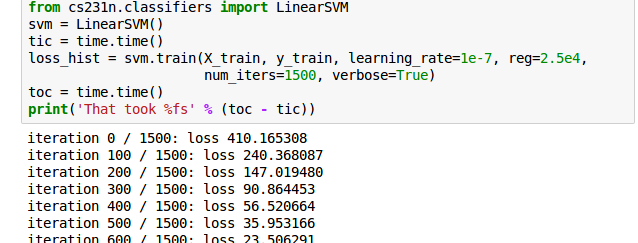
**Stochastic Gradient Descent:**

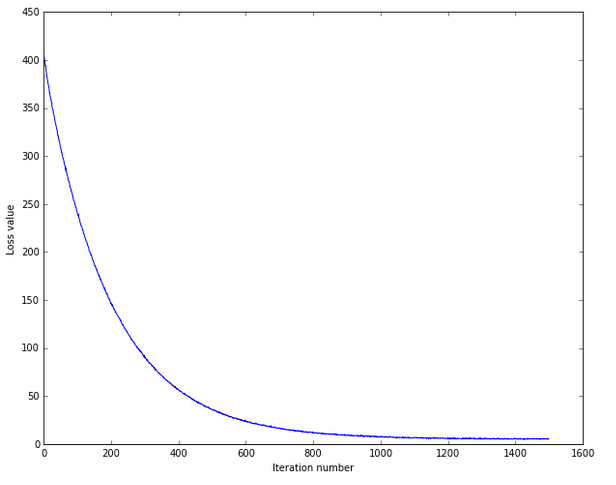








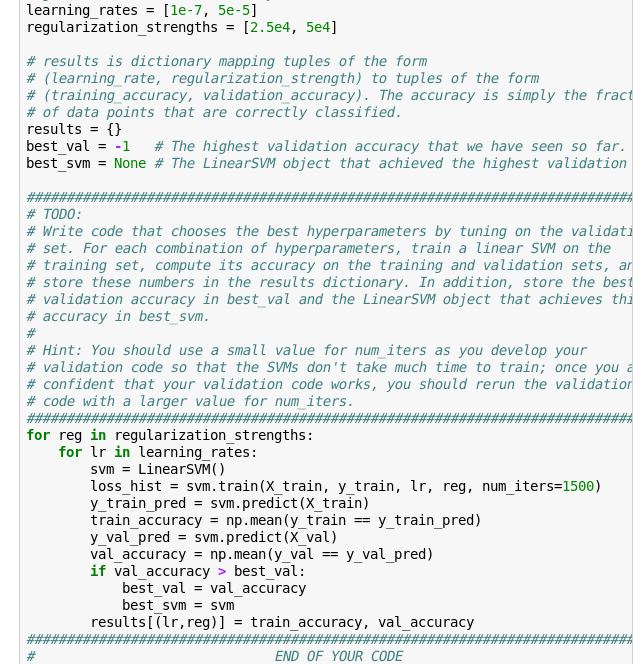


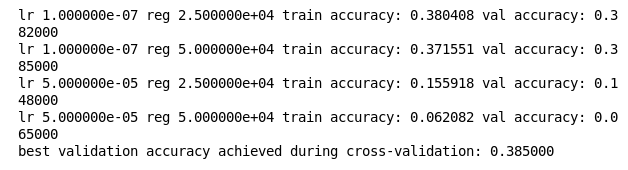


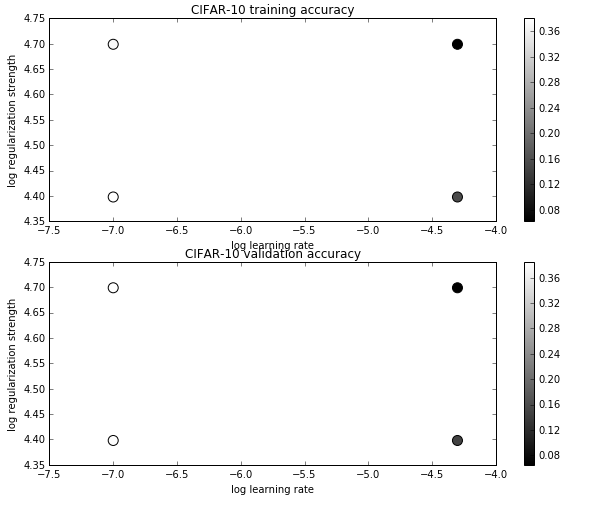
准确率:

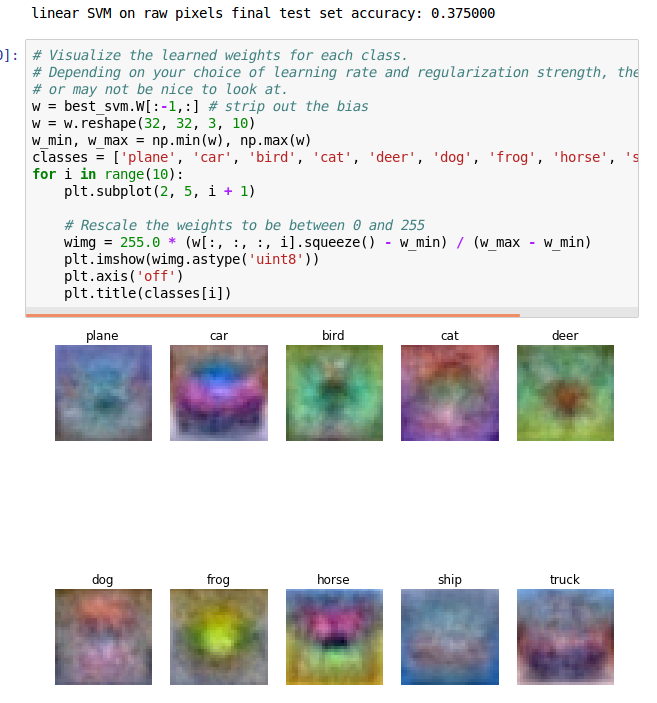


**调节超参数:**

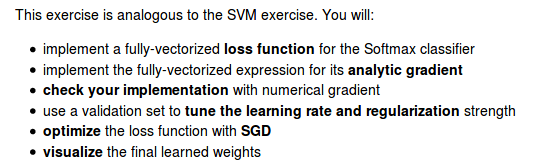


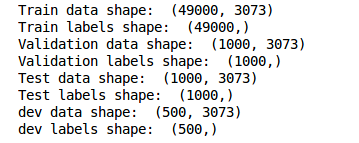


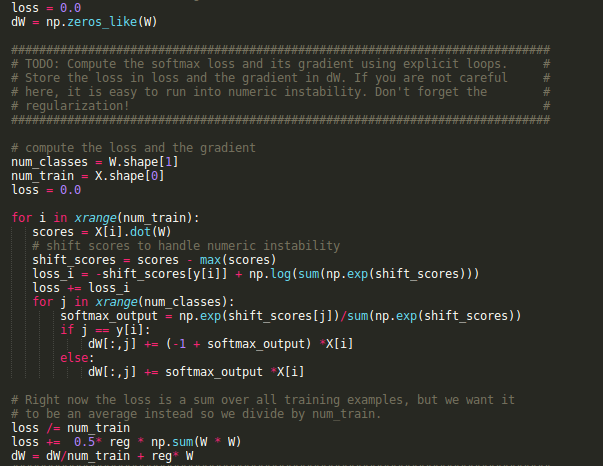




## Q3: Implement a Softmax classifier

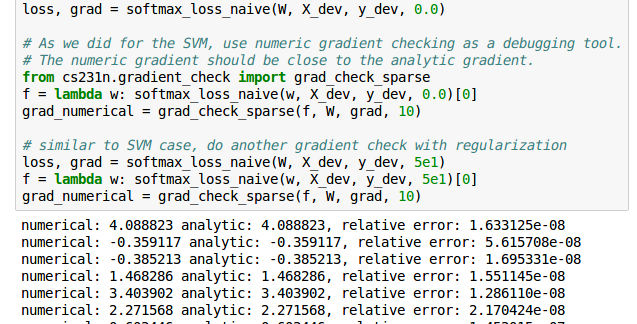








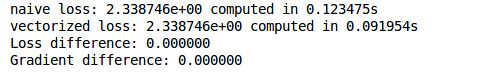
**梯度检查:**

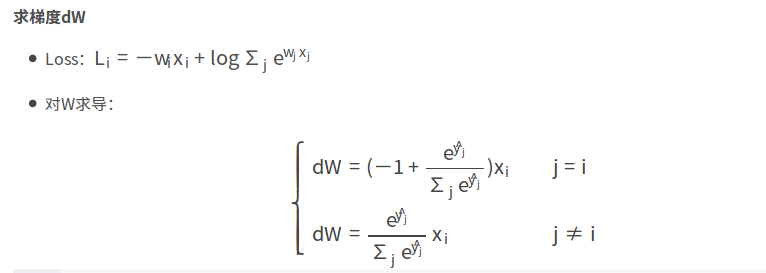


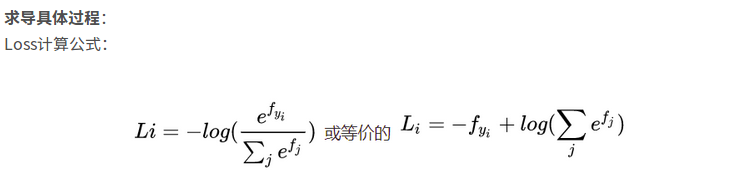
向量化:

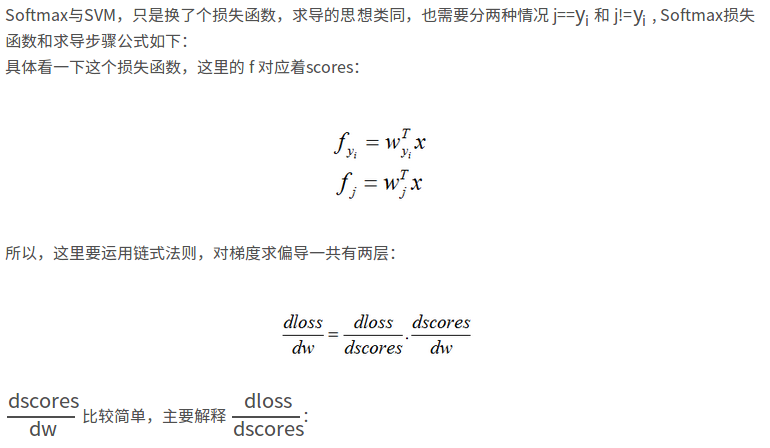


结果对比:



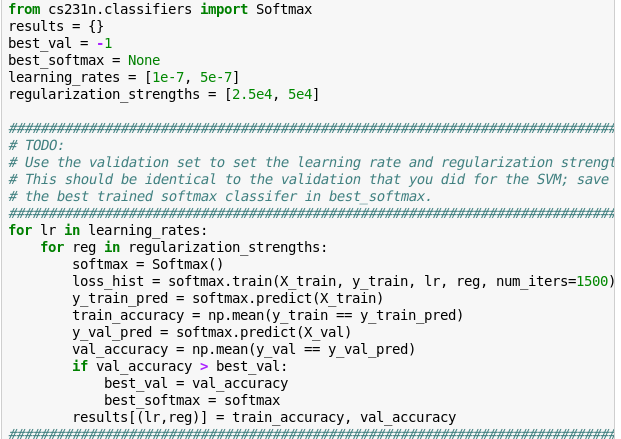


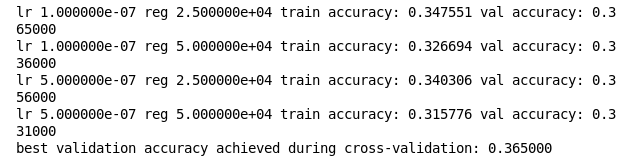






**超参数调优:**

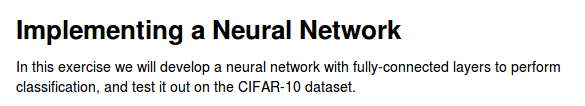


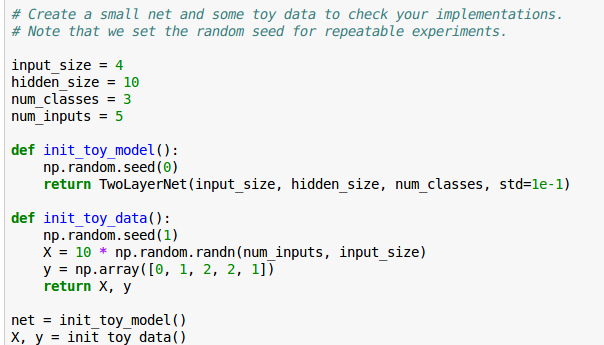


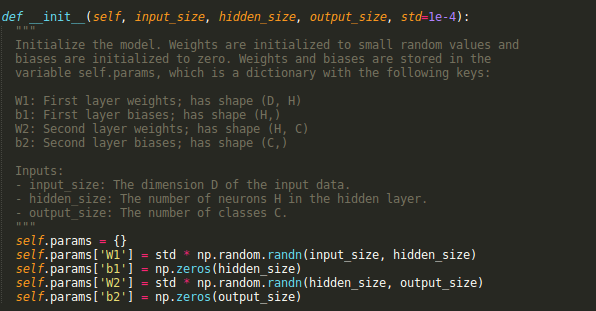
**可视化:**

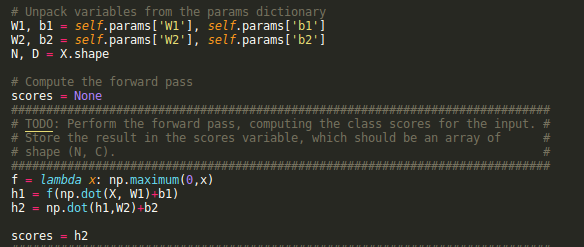


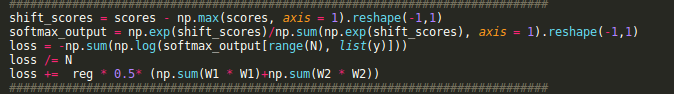
## Q4: Two-Layer Neural Network

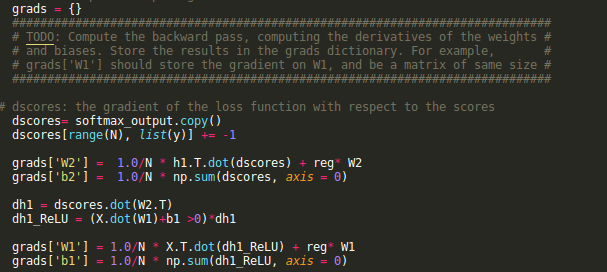


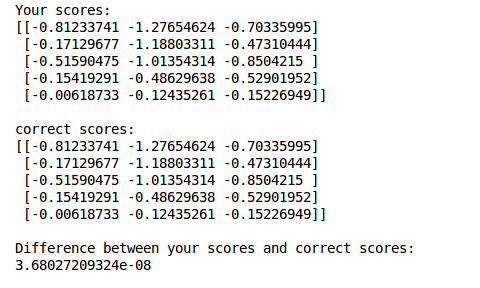








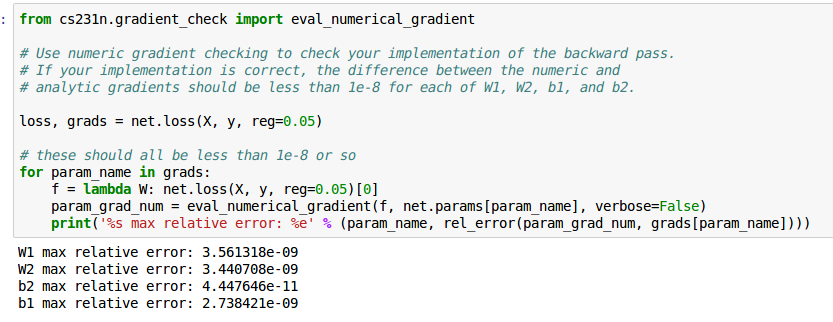




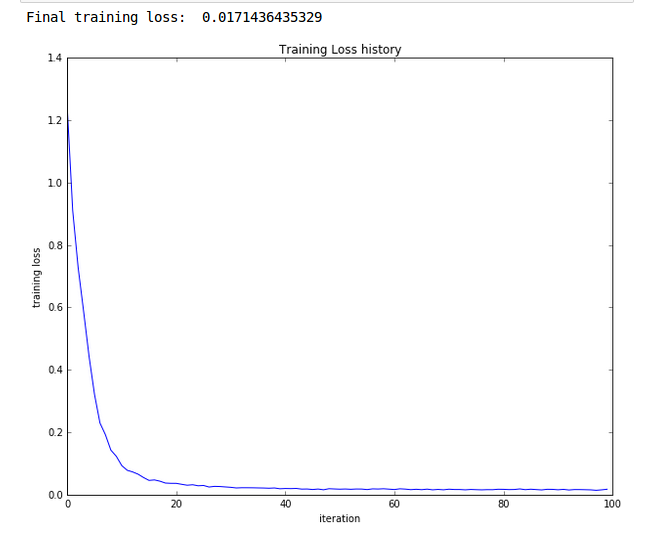
**Forward pass: compute loss:**

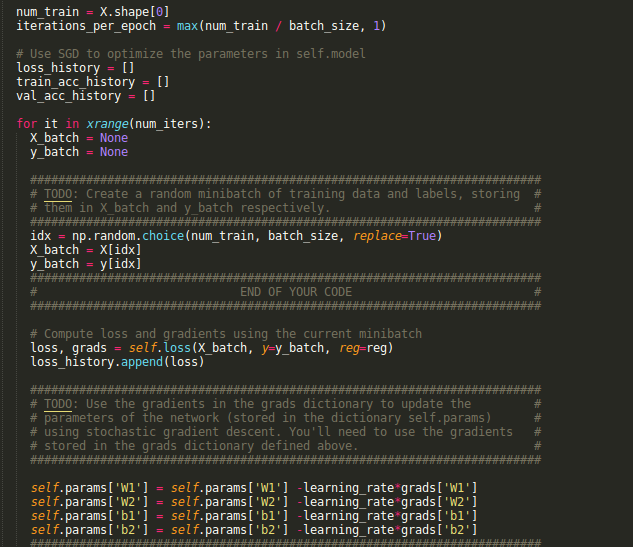


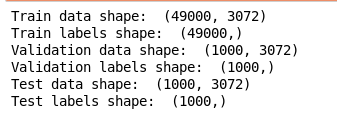
**Backward pass:**

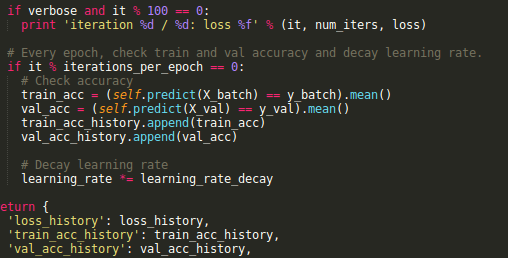


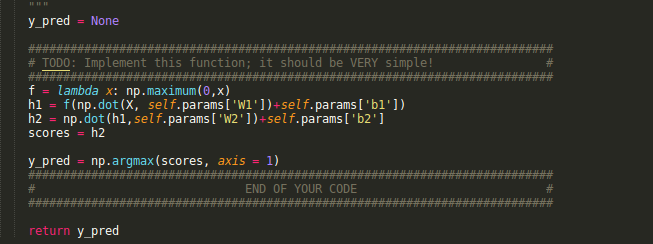
**Train the network:**

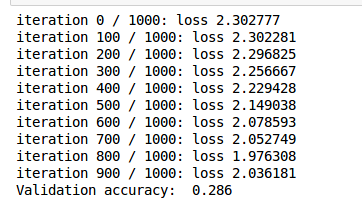


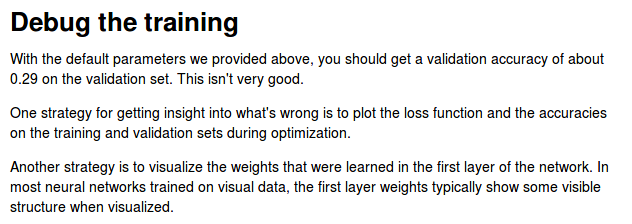


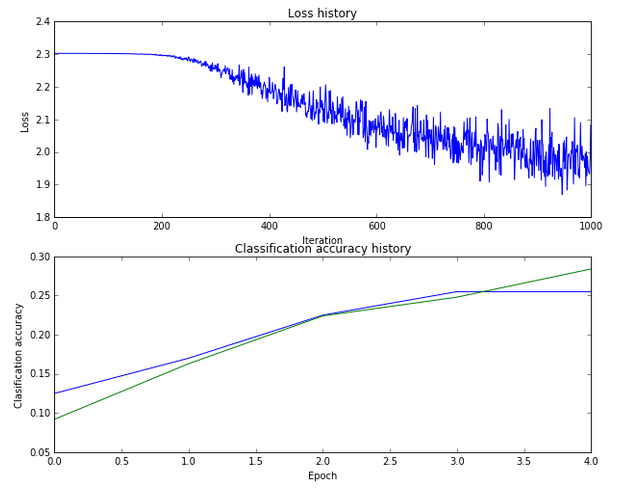


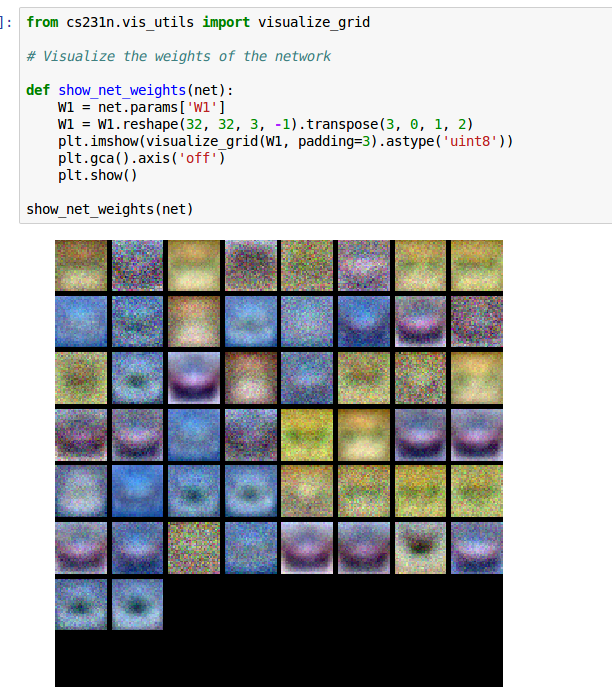




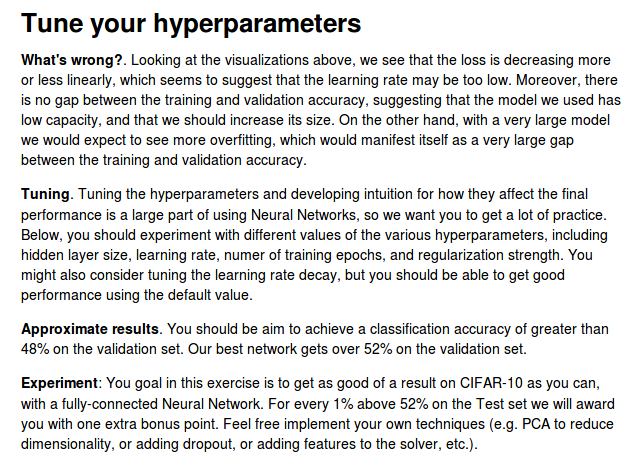


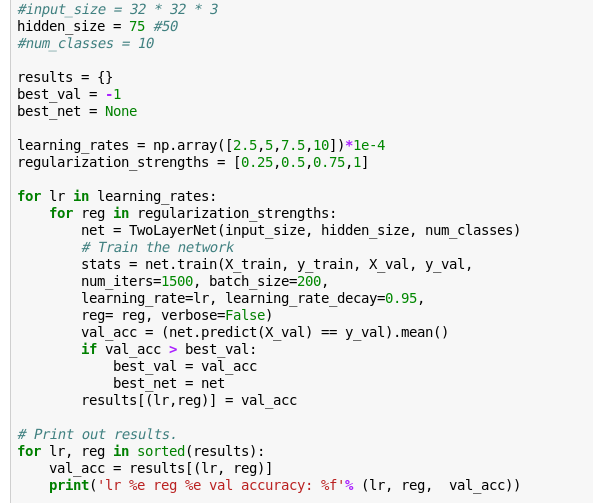


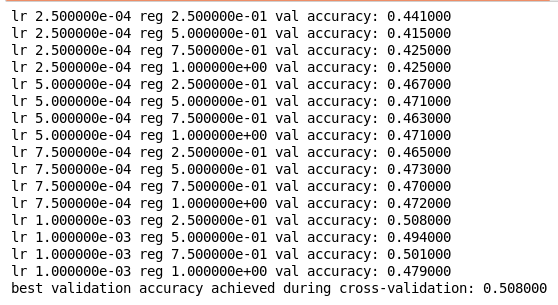


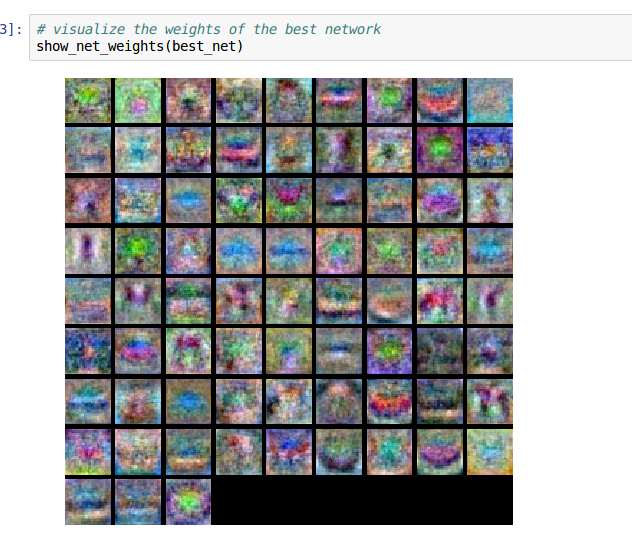


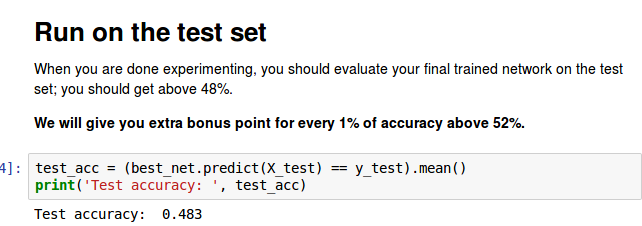
**超参数调优:**



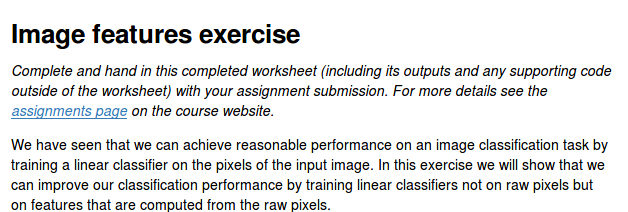








## Q5: Higher Level Representations: Image Features

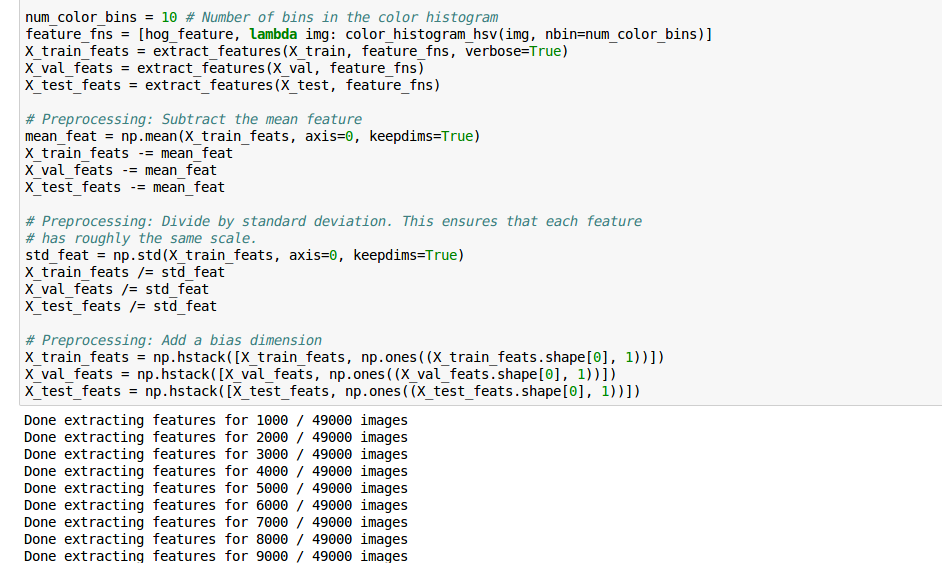


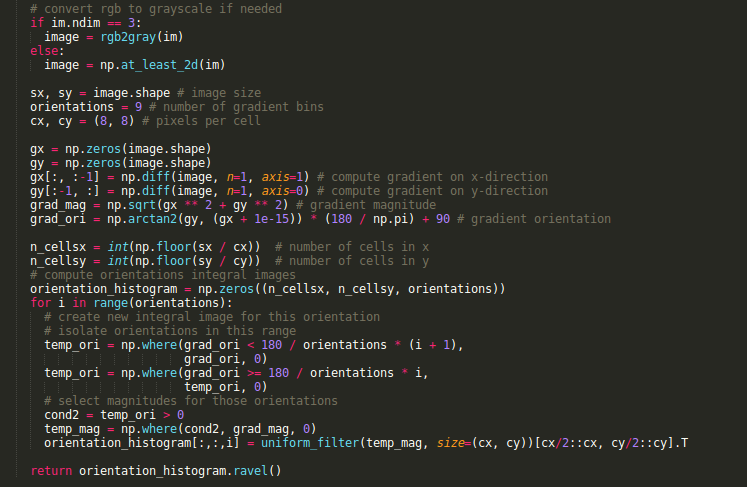
**Extract Features:**

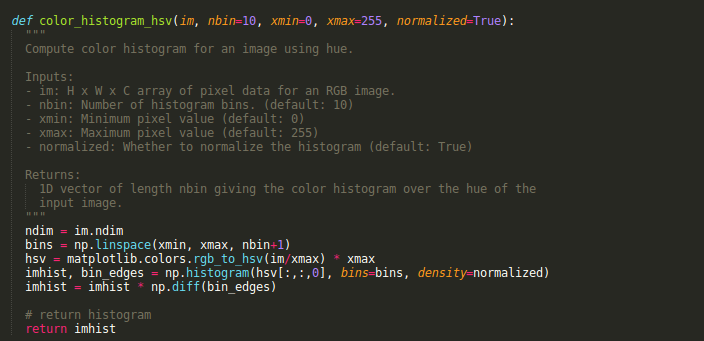
For each image we will compute a Histogram of Oriented Gradients (HOG) as well as a color histogram using the hue channel in HSV color space. We form our final feature vector for each image by concatenating the HOG and color histogram feature vectors.

Roughly speaking, HOG should capture the texture of the image while ignoring color information, and the color histogram represents the color of the input image while ignoring texture. As a result, we expect that using both together ought to work better than using either alone. Verifying this assumption would be a good thing to try for the bonus section.



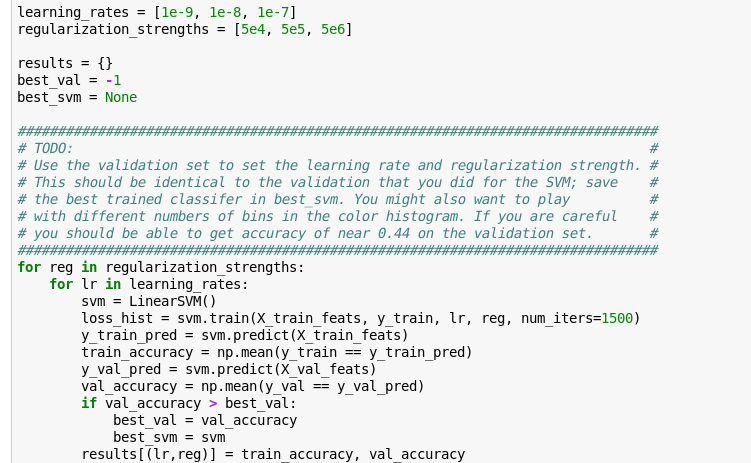


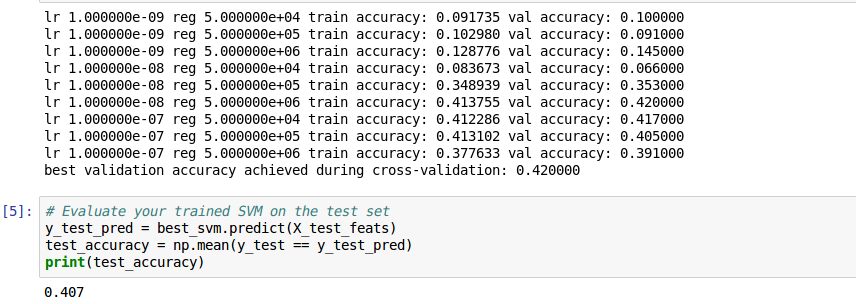




**Train SVM on features**

Using the multiclass SVM code developed earlier in the assignment, train SVMs on top of the features extracted above; this should achieve better results than training SVMs directly on top of raw pixels.

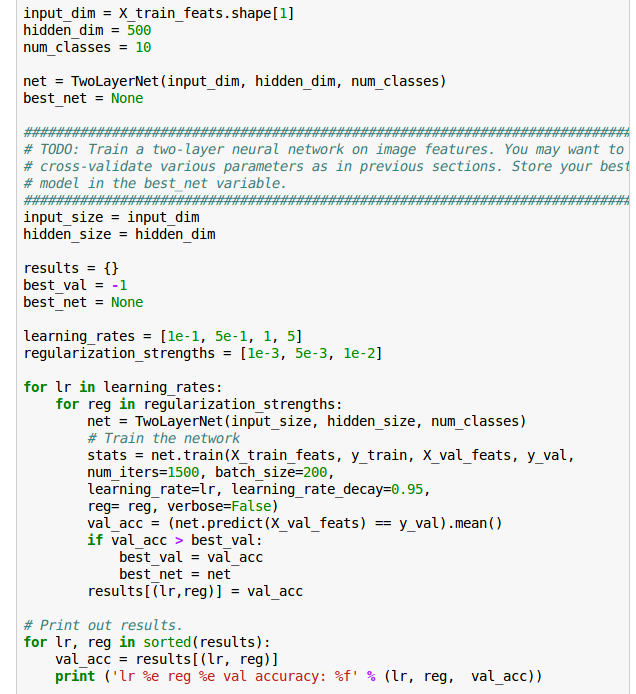


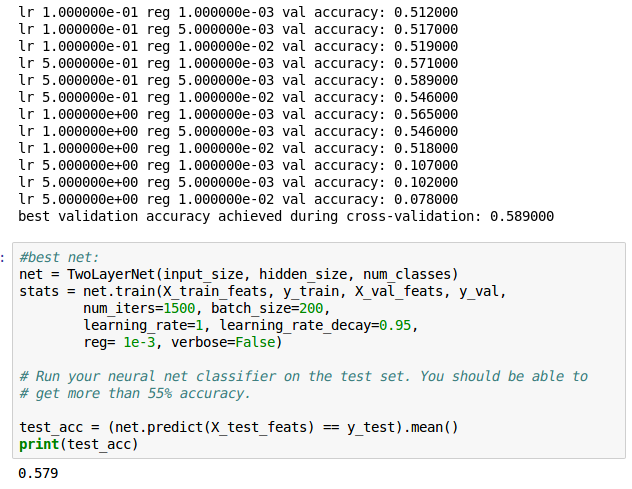


**Neural Network on image features**

Earlier in this assigment we saw that training a two-layer neural network on raw pixels achieved better classification performance than linear classifiers on raw pixels. In this notebook we have seen that linear classifiers on image features outperform linear classifiers on raw pixels.

For completeness, we should also try training a neural network on image features. This approach should outperform all previous approaches: you should easily be able to achieve over 55% classification accuracy on the test set; our best model achieves about 60% classification accuracy.





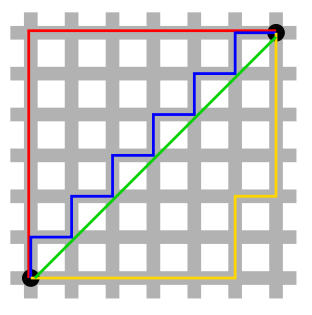
# 6. 对一些概念和函数的理解

## 6.1 曼哈顿距离和欧氏距离比较

出租车几何或曼哈顿距离（Manhattan Distance）是由十九世纪的赫尔曼·闵可夫斯基所创词汇 ，是种使用在几何度量空间的几何学用语，用以标明两个点在标准坐标系上的绝对轴距总和。图中红线代表曼哈顿距离，绿色代表欧氏距离，也就是直线距离，而蓝色和黄色代表等价的曼哈顿距离。

曼哈顿距离与欧氏距离：红 蓝和黄分别表示曼哈顿距离都拥有一样的长度12.

欧式距离也称为欧几里得距离。



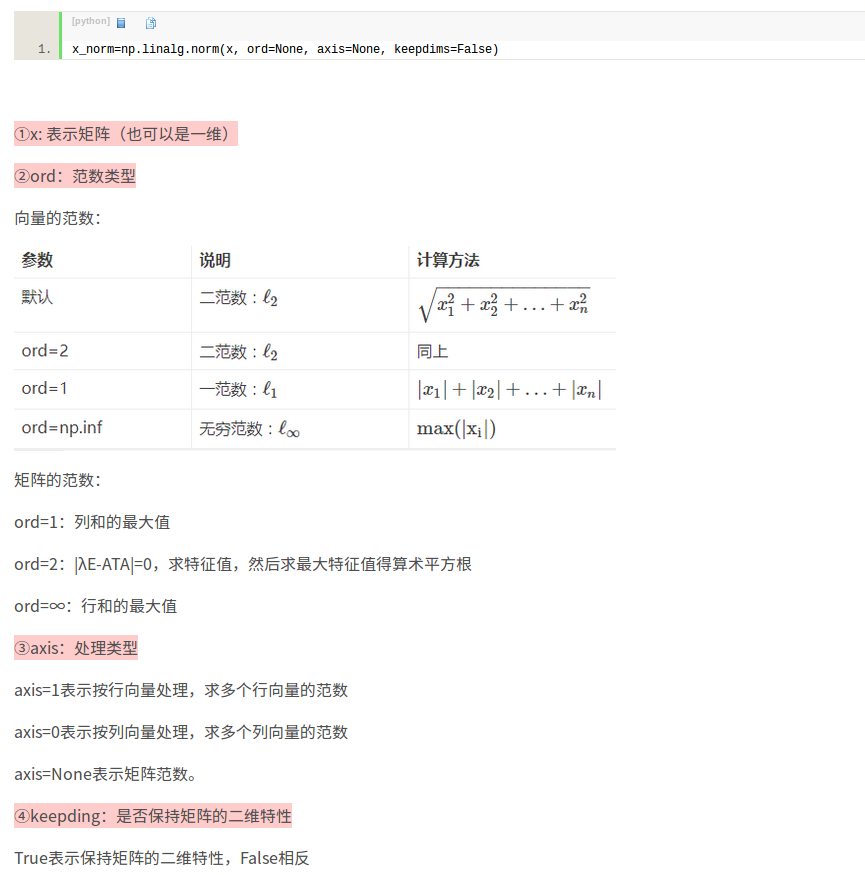
## 6.2 numpy的ravel() 和 flatten()函数

首先声明两者所要实现的功能是一致的（将多维数组降位一维）。这点从两个单词的意也可以看出来，ravel(散开，解开)，flatten（变平）。两者的区别在于返回拷贝（copy）还是返回视图（view），numpy.flatten()返回一份拷贝，对拷贝所做的修改不会影响（reflects）原始矩阵，而numpy.ravel()返回的是视图（view，也颇有几分C/C++引用reference的意味），会影响（reflects）原始矩阵。

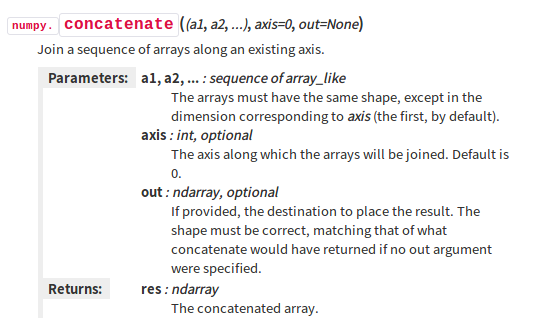


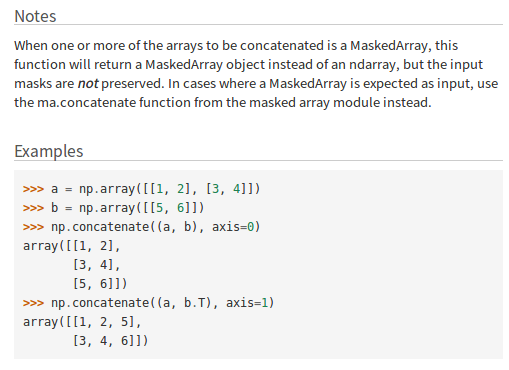
## 6.3 np.linalg.norm(求范数)

1. linalg=linear（线性）+algebra（代数），norm则表示范数。
2. 函数参数



## 6.4 Numpy.concatenate

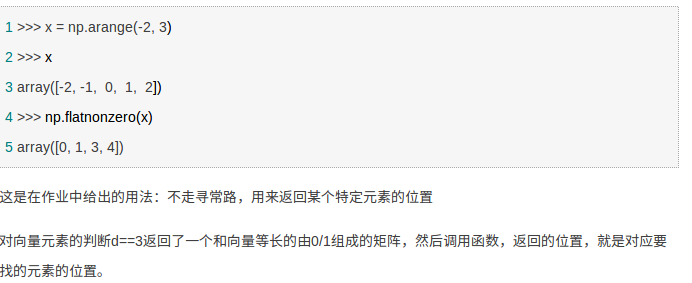




## 6.5 numpy.flatnonzero()

该函数输入一个矩阵，返回扁平化后矩阵中非零元素的位置（index）

这是官方文档给出的用法，非常正规，输入一个矩阵，返回了其中非零元素的位置



## 6.6 np.random.seed(0)

seed() 方法改变随机数生成器的种子，可以在调用其他随机模块函数之前调用此函数。

seed( ) 用于指定随机数生成时所用算法开始的整数值，如果使用相同的seed( )值，则每次生成的随即数都相同，如果不设置这个值，则系统根据时间来自己选择这个值，此时每次生成的随机数因时间差异而不同。

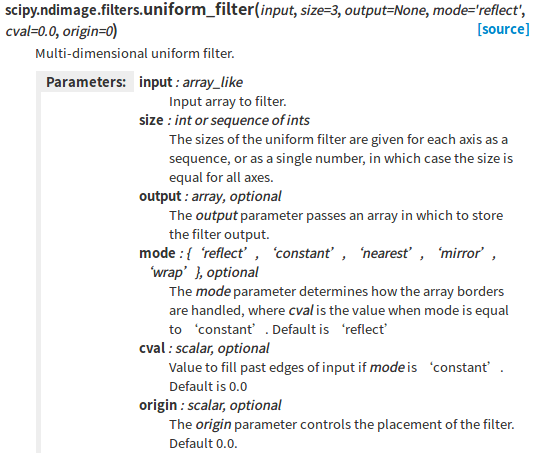
作用：使得随机数据可预测。



## 6.7 numpy的ravel() 和 flatten()函数

首先声明两者所要实现的功能是一致的（将多维数组降位一维）。这点从两个单词的意也可以看出来，ravel(散开，解开)，flatten（变平）。两者的区别在于返回拷贝（copy）还是返回视图（view），numpy.flatten()返回一份拷贝，对拷贝所做的修改不会影响（reflects）原始矩阵，而numpy.ravel()返回的是视图（view，也颇有几分C/C++引用reference的意味），会影响（reflects）原始矩阵。

## 6.8 scipy.ndimage.filters.uniform\_filter(均值滤波)



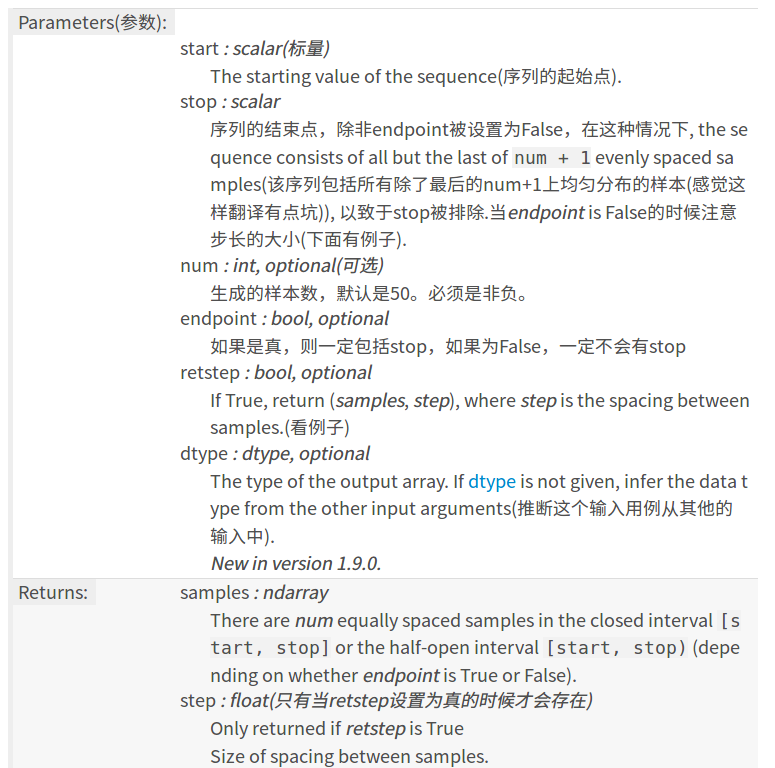
## 6.9 Numpy.linspace

numpy.linspace(start, stop, num=50, endpoint=True, retstep=False, dtype=None)

在指定的间隔内返回均匀间隔的数字。

返回num均匀分布的样本，在[start, stop]。

这个区间的端点可以任意的被排除在外。



## 6.10 np.where使用

根据条件生成新的数组

np.where 函数是三元表达式 x if condition else y的矢量化版本

result = np.where(cond,xarr,yarr)

当符合条件时是x，不符合是y，常用于根据一个数组产生另一个新的数组。

栗子：假设有一个随机数生成的矩阵，希望将所有正值替换为2，负值替换为-2

arr = np.random.randn(4,4)

arr

Out[51]:

array([[ 0.04150406, 1.27790573, -0.25917274, -1.25604622],

[ 0.8797799 , 1.84828821, -1.21709272, -0.41767649],

[-0.71758894, -0.70595454, 1.72330333, 0.18559916],

[-2.19529605, 2.11615947, -0.13563148, -1.41532576]])

np.where(arr>0,2,-2)

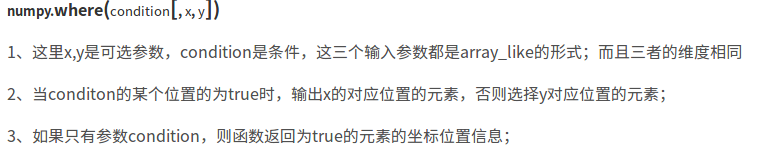
Out[52]:

array([[ 2, 2, -2, -2],

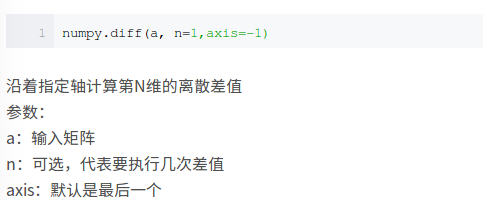
[ 2, 2, -2, -2],

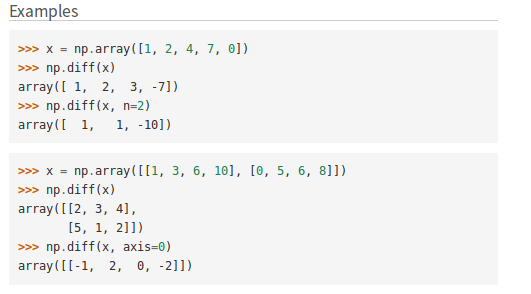
[-2, -2, 2, 2],

[-2, 2, -2, -2]])



## 6.11 numpy diff函数





## 6.12 Np.histogram

官方文档:

https://docs.scipy.org/doc/numpy/reference/generated/numpy.histogram.html

