CS231n课程笔记翻译：线性分类笔记

原文如下

内容列表：

线性分类

上一篇笔记介绍了图像分类问题。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。

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$x_i \in R^D$，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$y_i \in 1....K$。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：$f : R^D \rightarrow R^K$，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：

$$f(x_i, W, b) = Wx_i + b$$

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。

需要注意的几点：

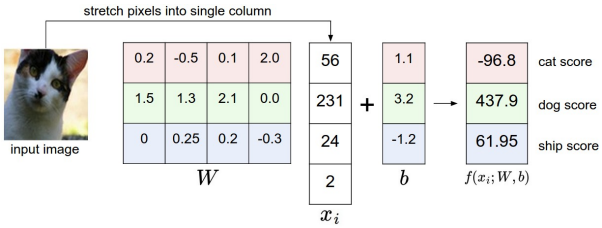
- 首先，一个单独的矩阵乘法Wx_i就高效地并行评估10个不同的分类器（每个分类器针对一个分类），其中每个类的分类器就是W的一个行向量。
- 注意我们认为输入数据(x_i, y_i)是给定且不可改变的，但参数**W**和**b**是可控制改变的。我们的目标就是通过设置这些参数，使得计算出来的分类分值情况和训练集中图像数据的真实类别标签相符。在接下来的课程中，我们将详细介绍如何做到这一点，但是目前只需要直观地让正确分类的分值比错误分类的分值高即可。
- 该方法的一个优势是训练数据是用来学习到参数**W**和**b**的，一旦训练完成，训练数据就可以丢弃，留下学习到的参数即可。这是因为一个测试图像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。

> *预告：卷积神经网络映射图像像素值到分类分值的方法和上面一样，但是映射(f)就要复杂多了，其包含的参数也更多。*

理解线性分类器

线性分类器计算图像中3个颜色通道中所有像素的值与权重的矩阵乘，从而得到分类分值。根据我们对权重设置的值，对于图像中的某些位置的某些颜色，函数表现出喜好或者厌恶（根据每个权重的符号而定）。举个例子，可以想象“船”分类就是被大量的蓝色所包围（对应的就是水）。那么“船”分类器在蓝色通道上的权重就有很多的正权重（它们的出现提高了“船”分类的分值），而在绿色和红色通道上的权重为负的就比较多（它们的出现降低了“船”分类的分值）。

——



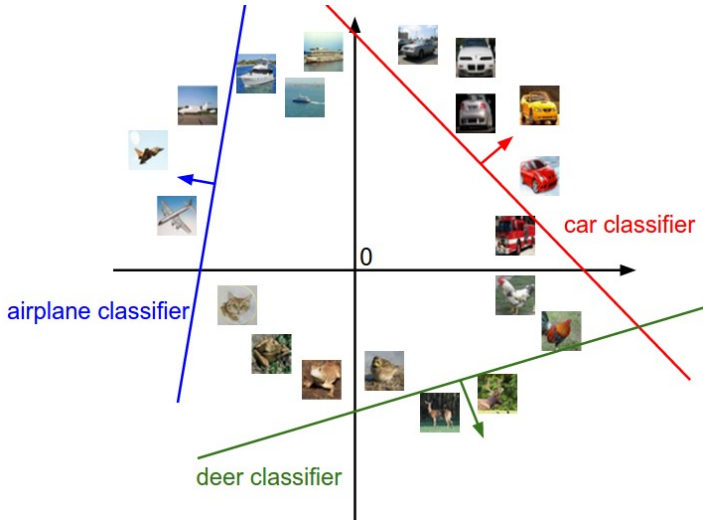
一个将图像映射到分类分值的例子。为了便于可视化，假设图像只有4个像素（都是黑白像素，这里不考虑RGB通道），有3个分类（红色代表猫，绿色代表狗，蓝色代表船，注意，这里的红、绿和蓝3种颜色仅代表分类，和RGB通道没有关系）。首先将图像像素拉伸为一个列向量，与W进行矩阵乘，然后得到各个分类的分值。需要注意的是，这个W一点也不好：猫分类的分值非常低。从上图来看，算法倒是觉得这个图像是一只狗。

——

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。

——

从上面可以看到，**W**的每一行都是一个分类类别的分类器。对于这些数字的几何解释是：如果改变其中一行的数字，会看见分类器在空间中对应的直线开始向着不同方向旋转。而偏差**b**，则允许分类器对应的直线平移。需要注意的是，如果没有偏差，无论权重如何，在$\boldsymbol{x_i} = \mathbf{0}$时分类分值始终为0。这样所有分类器的线都不得不穿过原点。

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。

——

将课程进度快进一点。这里展示的是以CIFAR-10为训练集，学习结束后的权重的例子。注意，船的模板如期望的那样有很多蓝色像素。如果图像是一艘船行驶在大海上，那么这个模板利用内积计算图像将给出很高的分数。

——

可以看到马的模板看起来似乎是两个头的马，这是因为训练集中的马的图像中马头朝向各有左右造成的。线性分类器将这两种情况融合到一起了。类似的，汽车的模板看起来也是将几个不同的模型融合到了一个模板中，并以此来分辨不同方向不同颜色的汽车。这个模板上的车是红色的，这是因为CIFAR-10中训练集的车大多是红色的。线性分类器对于不同颜色的车的分类能力是很弱的，但是后面可以看到神经网络是可以完成这一任务的。神经网络可以在它的隐藏层中实现中间神经元来探测不同种类的车（比如绿色车头向左，蓝色车头向前等）。而下一层的神经元通过计算不同的汽车探测器的权重和，将这些合并为一个更精确的汽车分类分值。

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$\boldsymbol{W}$和$\boldsymbol{b}$合二为一。回忆一下，分类评分函数定义为：

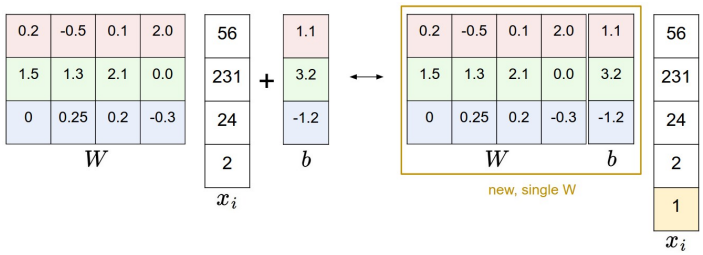
$$f(x_i, W, b) = Wx_i + b$$

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：

$$f(x_i, W) = Wx_i$$

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，等我们理解了梯度下降后再来详细解释。

损失函数 Loss function

在上一节定义了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）。

回到之前那张猫的图像分类例子，它有针对“猫”，“狗”，“船”三个类别的分数。我们看到例子中权重值非常差，因为猫分类的得分非常低（-96.8），而狗（437.9）和船（61.95）比较高。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。

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Δ。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：$s_j = f(x_i, W)_j$。针对第i个数据的多类SVM的损失函数定义如下：

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + \Delta)$$

举例：用一个例子演示公式是如何计算的。假设有3个分类，并且得到了分值$s = [13, -7, 11]$。其中第一个类别是正确类别，即$y_i = 0$。同时假设Δ是10（后面会详细介绍该超参数）。上面的公式是将所有不正确分类（$j \neq y_i$）加起来，所以我们得到两个部分：

可以看到第一个部分结果是0，这是因为[-7-13+10]得到的是负数，经过$max(0, -)$函数处理后得到0。这一对类别分数和标签的损失值是0，这是因为正确分类的得分13与错误分类的得分-7的差为20，高于边界值10。而SVM只关心差距至少要大于10，更大的差值还是算作损失值为0。第二个部分计算[11-13+10]得到8。虽然正确分类的得分比不正确分类的得分要高（13>11），但是比10的边界值还是小了，分差只有2，这就是为什么损失值等于8。简而言之，SVM的损失函数想要正确分类类别y_i的分数比不正确类别分数高，而且至少要高Δ。如果不满足这点，就开始计算损失值。

那么在这次的模型中，我们面对的是线性评分函数（$f(x_i, W) = Wx_i$），所以我们可以将损失函数的公式稍微改写一下：

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

其中w_j是权重的W第j行，被变形为列向量。然而，一旦开始考虑更复杂的评分函数f公式，这样做就不是必须的了。

在结束这一小节前，还必须提一下的属于是关于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$\lambda > 1$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：

$$\sum_k \sum_l W_{k,l}^2$$

上面的表达式中，将W中所有元素平方后求和。注意正则化函数不是数据的函数，仅基于权重。包含正则化惩罚后，就能够给出完整的多类SVM损失函数了，它由两个部分组成：**数据损失（data loss）**，即所有样例的的平均损失L_i，以及**正则化损失（regularization loss）**。完整公式如下所示：

$$L = \underbrace{\frac{1}{N}\sum_i L_i}_{data\ loss} + \underbrace{\lambda R(W)}_{regularization\ loss}$$

将其展开完整公式是：

$$L = \frac{1}{N}\sum_i\sum_{j\neq y_i}[max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + \Delta)] + \lambda\sum_k\sum_l W_{k,l}^2$$

其中，N是训练集的数据量。现在正则化惩罚添加到了损失函数里面，并用超参数λ来计算其权重。该超参数无法简单确定，需要通过交叉验证来获取。

除了上述理由外，引入正则化惩罚还带来很多良好的性质，这些性质大多会在后续章节介绍。比如引入了L2惩罚后，SVM们就有了**最大边界（max margin）**这一良好性质。（如果感兴趣，可以查看CS229课程）。

其中最好的性质就是对大数值权重进行惩罚，可以提升其泛化能力，因为这就意味着没有哪个维度能够独自对于整体分值有过大的影响。举个例子，假设输入向量$x = [1,1,1,1]$，两个权重向量$w_1 = [1,0,0,0]$，$w_2 = [0.25, 0.25, 0.25, 0.25]$。那么$w_1^T x = w_2^T = 1$,两个权重向量都得到同样的内积，但是$w_1$的L2惩罚是1.0，而$w_2$的L2惩罚是0.25。因此，根据L2惩罚来看，$w_2$更好，因为它的正则化损失更小。从直观上来看，这是因为w_2的权重值更小且更分散。既然L2惩罚倾向于更小更分散的权重向量，这就会鼓励分类器最终将所有维度上的特征都用起来，而不是强烈依赖其中少数几个维度。在后面的课程中可以看到，这一效果将会提升分类器的泛化能力，并避免*过拟合*。

需要注意的是，和权重不同，偏差没有这样的效果，因为它们并不控制输入维度上的影响强度。因此通常只对权重W正则化，而不正则化偏差b。在实际操作中，可发现这一操作的影响可忽略不计。最后，因为正则化惩罚的存在，不可能在所有的例子中得到0的损失值，这是因为只有当$W = 0$的特殊情况下，才能得到损失值为0。

代码：下面是一个无正则化部分的损失函数的Python实现，有非向量化和半向量化两个形式：

```
def L_i(x, y, W):
  """
  unvectorized version. Compute the multiclass svm loss for a single example (x,y)
  - x is a column vector representing an image (e.g. 3073 x 1 in CIFAR-10)
    with an appended bias dimension in the 3073-rd position (i.e. bias trick)
  - y is an integer giving index of correct class (e.g. between 0 and 9 in CIFAR-10)
  - W is the weight matrix (e.g. 10 x 3073 in CIFAR-10)
  """
  delta = 1.0 # see notes about delta later in this section
  scores = W.dot(x) # scores becomes of size 10 x 1, the scores for each class
  correct_class_score = scores[y]
  D = W.shape[0] # number of classes, e.g. 10
  loss_i = 0.0
  for j in xrange(D): # iterate over all wrong classes
    if j == y:
      # skip for the true class to only loop over incorrect classes
      continue
    # accumulate loss for the i-th example
    loss_i += max(0, scores[j] - correct_class_score + delta)
  return loss_i

def L_i_vectorized(x, y, W):
  """
  A faster half-vectorized implementation. half-vectorized
  refers to the fact that for a single example the implementation contains
  no for loops, but there is still one loop over the examples (outside this function)
  """
  delta = 1.0
  scores = W.dot(x)
  # compute the margins for all classes in one vector operation
  margins = np.maximum(0, scores - scores[y] + delta)
  # on y-th position scores[y] - scores[y] canceled and gave delta. We want
  # to ignore the y-th position and only consider margin on max wrong class
  margins[y] = 0
  loss_i = np.sum(margins)
  return loss_i

def L(X, y, W):
  """
  fully-vectorized implementation :
  - X holds all the training examples as columns (e.g. 3073 x 50,000 in CIFAR-10)
  - y is array of integers specifying correct class (e.g. 50,000-D array)
  - W are weights (e.g. 10 x 3073)
  """
  # evaluate loss over all examples in X without using any for loops
  # left as exercise to reader in the assignment
```

在本小节的学习中，一定要记得SVM损失采取了一种特殊的方法，使得能够衡量对于训练数据预测分类和实际分类标签的一致性。还有，对训练集中数据做出准确分类预测和让损失值最小化这两件事是等价的。

> 接下来要做的，就是找到能够使损失值最小化的权重了。

实际考虑

设置Delta：你可能注意到上面的内容对超参数Δ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$\Delta = 1.0$都是安全的。超参数Δ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$\Delta = 1$或$\Delta = 100$）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（通过正则化强度λ来控制）。

与二元支持向量机（Binary Support Vector Machine）的关系：在学习本课程前，你可能对于二元支持向量机有些经验，它对于第i个数据的损失计算公式是：

其中，C是一个超参数，并且$y_i \in \{-1, 1\}$。可以认为本章节介绍的SVM公式包含了上述公式，上述公式是多类支持向量机公式只有两个分类类别的特例。也就是说，如果我们要分类的类别只有两个，那么公式就化为二元SVM公式。这个公式中的C和多类SVM公式中的λ都控制着同样的权衡，而且它们之间的关系是$C \propto \frac{1}{\lambda}$

备注：在初始形式中进行最优化。如果在本课程之前学习过SVM，那么对kernels，duals，SMO算法等将有所耳闻。在本课程（主要是神经网络相关）中，损失函数的最优化的始终在非限制初始形式下进行。很多这些损失函数从技术上来说是不可微的（比如当$x = y$时，$max(x, y)$函数就不可微分），但是在实际操作中并不存在问题，因为通常可以使用次梯度。

备注：其他多类SVM公式。需要指出的是，本课中展示的多类SVM只是多种SVM公式中的一种。另一种常用的公式是*One-Vs-All*（OVA）SVM，它针对每个类和其他类训练一个独立的二元分类器。还有另一种更少用的叫做*All-Vs-All*（AVA）策略。我们的公式是按照Weston and Watkins 1999 (pdf)版本，比OVA性能更强（在构建有一个多类数据集的情况下，这个版本可以在损失值上取到0，而OVA就不行。感兴趣的话在论文中查阅细节）。最后一个需要知道的公式是Structured SVM，它将正确分类的分类分值和非正确分类中的最高分值的边界最大化。理解这些公式的差异超出了本课程的范围。本课程笔记介绍的版本可以在实践中安全使用，而被论证为最简单的OVA策略在实践中看起来也能工作的同样出色（在 Rikin等人2004年的论文In Defense of One-Vs-All Classification (pdf)中可查）。

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：

$$Li = -log(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}) \text{ 或等价的 } L_i = -f_{y_i} + log(\sum_j e^{f_j})$$

在上式中，使用f_j来表示分类评分向量f中的第j个元素。和之前一样，整个数据集的损失值是数据集中所有样本数据的损失值L_i之和。其中函数$f_j(z) = \frac{e^{z_j}}{\sum_k e^{z_k}}$被称作**softmax 函数**：其输入值是一个向量，向量中元素为任意实数的评分值（z中的），函数对其进行压缩，输出一个向量，其中每个元素值在0到1之间，且所有元素之和为1。所以，包含softmax函数的完整交叉熵损失看起唬人，实际上还是比较容易理解的。

信息理论视角：在"真实"分布p和估计分布q之间的*交叉熵*定义如下：

因此，Softmax分类器所做的就是最小化在估计分类概率（就是上面的$e^{f_{y_i}} / \sum_j e^{f_j}$）和“真实”分布之间的交叉熵，在这个解释中，“真实”分布就是所有概率密度都分布在正确的类别上（比如：$p = [0, \ldots 1, \ldots, 0]$中在$y_i$的位置就有一个单独的1）。还有，既然交叉熵可以写成熵和相对熵（Kullback-Leibler divergence）$H(p, q) = H(p) + D_{KL}(p||q)$，并且delta函数$p$的熵是0，那么就能等价的看做是对两个分布之间的相对熵做最小化操作。换句话说，交叉熵损失函数“想要”预测分布的所有概率密度都在正确分类上。

> *译者注：Kullback-Leibler差异（Kullback-Leibler Divergence）也叫做相对熵（Relative Entropy），它衡量的是相同事件空间里的两个概率分布的差异情况。*

概率论解释：先看下面的公式：

可以解释为是给定图像数据x_i为参数，分配给正确分类标签y_i的归一化概率。为了理解这点，请回忆一下Softmax分类器将输出向量f中的评分值解释为没有归一化的对数概率。那么以这些数值做指数函数的幂就得到了没有归一化的概率，而除法操作则对数据进行了归一化处理，使得这些概率的和为1。从概率论的角度来理解，我们就是在最小化正确分类的负对数概率，这可以看做是在进行*最大似然估计*（MLE）。该解释的另一个好处是，损失函数中的正则化部分$R(W)$可以被看做是权重矩阵W的高斯先验，这里进行的是最大后验估计（MAP）而不是最大似然估计。提及这些解释只是为了让读者形成直观的印象，具体细节就超过本课程范围了。

实操事项：数值稳定。编程实现softmax函数计算的时候，中间项$e^{f_{y_i}}$和$\sum_j e^{f_j}$因为存在指数函数，所以数值可能非常大。除以大数值可能导致数值计算的不稳定，所以学会使用归一化技巧非常重要。如果在分式的分子和分母都乘以一个常数C，并把它变换到求和之中，就能得到一个从数学上等价的公式：

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$logC = -max_j f_j$。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。代码实现如下：

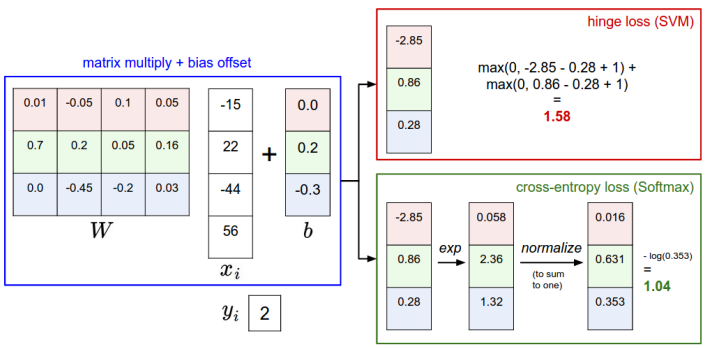
```
f = np.array([123, 456, 789]) # 例子中有3个分类，每个评分的数值都很大
p = np.exp(f) / np.sum(np.exp(f)) # 不妙：数值问题，可能导致数值爆炸

# 那么将f中的值平移到最大值为0：
f -= np.max(f) # f becomes [-666, -333, 0]
p = np.exp(f) / np.sum(np.exp(f)) # 现在OK了，将给出正确结果
```

让人迷惑的命名规则：精确地说，SVM分类器使用的是*折叶损失（hinge loss）*，有时候又被称为*最大边界损失（max-margin loss）*。Softmax分类器使用的是*交叉熵损失（corss-entropy loss）*。Softmax分类器的命名是从*softmax函数*那里得来的，softmax函数将原始分类评分变成正的归一化数值，所有数值和为1，这样处理后交叉熵损失才能应用。注意从技术上说“softmax损失（softmax loss）”是没有意义的，因为softmax只是一个压缩数值的函数。但是在这个说法常常被用来做简称。

SVM和Softmax的比较

下图有助于区分这 Softmax和SVM这两种分类器：



SVM和Softmax的比较

下图有助于区分这 Softmax和SVM这两种分类器：

——

针对一个数据点，SVM和Softmax分类器的不同处理方式的例子。两个分类器都计算了同样的分值向量**f**（本节中是通过矩阵乘来实现）。不同之处在于对**f**中分值的解释：SVM分类器将它们看做是分类评分，它的损失函数鼓励正确的分类（本例中是蓝色的类别2）的分值比其他分类的分值高出至少一个边界值。Softmax分类器将这些数值看做是每个分类没有归一化的**对数概率**，鼓励正确分类的归一化的对数概率变高，其余的变低。SVM的最终的损失值是1.58，Softmax的最终的损失值是0.452，但要注意这两个数值没有可比性。只在给定同样数据，在同样的分类器的损失值计算中，它们才有意义。

——

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。举个例子，针对给出的图像，SVM分类器可能给你的是一个[12.5, 0.6, -23.0]对应分类"猫"，"狗"，"船"。而softmax分类器可以计算出这三个标签的"可能性"是[0.9, 0.09, 0.01]，这就让你能看出对于不同分类准确性的把握。为什么我们要在"可能性"上面打引号呢？这是因为可能性分布的集中或离散程度是由正则化参数λ直接决定的，λ是你能直接控制的一个输入参数。举个例子，假设3个分类的原始分数是[1, -2, 0]，那么softmax函数就会计算：

现在，如果正则化参数λ更大，那么权重W就会被惩罚的更多，然后他的权重数值就会更小。这样算出来的分数也会更小，假设小了一半吧[0.5, -1, 0]，那么softmax函数的计算就是：

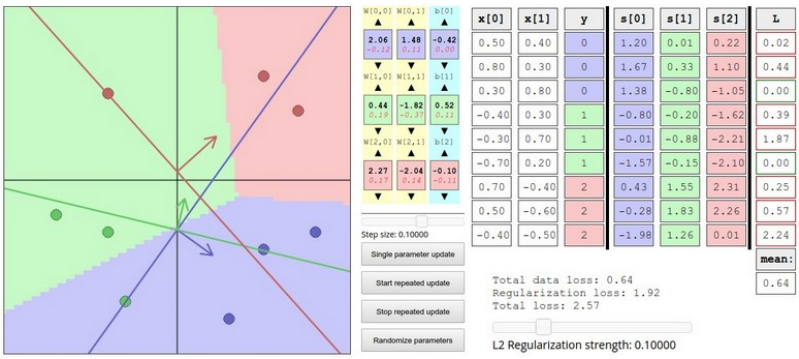
现在看起来，概率的分布就更加分散了。还有，随着正则化参数λ不断增强，权重数值会越来越小，最后输出的概率会接近于均匀分布。这就是说，softmax分类器算出来的概率最好是看成一种对于分类正确性的自信。和SVM一样，数字间相互比较得出的大小顺序是可以解释的，但其绝对值则难以直观解释。

在实际使用中，SVM和Softmax经常是相似的：通常说来，两种分类器的表现差别很小，不同的人对于哪个分类器更好有不同的看法。相对于Softmax分类器，SVM更加"局部目标化（local objective）"，这既可以看做是一个特性，也可以看做是一个劣势。考虑一个评分是[10, -2, 3]的数据，其中第一个分类是正确的。那么一个SVM（$\Delta = 1$）会看到正确分类相较于不正确分类，已经得到了比边界值还要高的分数，它就会认为损失值是0。SVM对于数字个体的细节是不关心的：如果分数是[10, -100, -100]或者[10, 9, 9]，对于SVM来说没设么不同，只要满足超过边界值等于1，那么损失值就等于0。

对于softmax分类器，情况则不同。对于[10, 9, 9]来说，计算出的损失值就远远高于[10, -100, -100]的。换句话来说，softmax分类器对于分数是永远不会满意的：正确分类总能得到更高的可能性，错误分类总能得到更低的可能性，损失值总是能够更小。但是，SVM只要边界值被满足了就满意了，不会超过限制去细微地操作具体分数。这可以被看做是SVM的一种特性。举例说来，一个汽车的分类器应该把他的大量精力放在如何分辨小轿车和大卡车上，而不应该纠结于如何与青蛙进行区分，因为区分青蛙得到的评分已经足够低了。

交互式的网页Demo

——



我们实现了一个交互式的网页原型，来帮助读者直观地理解线性分类器。原型将损失函数进行可视化，画面表现的是对于2维数据的3种类别的分类。原型在课程进度上稍微超前，展现了最优化的内容，最优化将在下一节课讨论。

——

小结

总结如下：

- 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。
- 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。
- 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。
- 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，将在下节课中进行介绍。

拓展阅读

下面的内容读者可根据兴趣选择性阅读。

- **Deep Learning using Linear Support Vector Machines**一文的作者是Tang Charlie，论文写于2013年，展示了一些L2SVM比Softmax表现更出色的结果。

线性分类笔记全文翻译完毕。

> 译自斯坦福CS231n课程笔记**Linear Classification Note**，课程教师**Andrej Karpathy**授权翻译。本篇教程由杜客翻译完成，堃堃进行校对修改
>
> 知乎地址：（上，中，下）
>
> https://zhuanlan.zhihu.com/p/20918580?refer=intelligentunit
>
> https://zhuanlan.zhihu.com/p/20945670?refer=intelligentunit
>
> https://zhuanlan.zhihu.com/p/21102293?refer=intelligentunit