【译】cs231n第二课：线性分类 - SVM与Softmax

2016-07-15 | 76

> 本来是要学习cs224d的，但cs224d要求学几节cs231n的课程作为预备课程，故有如此翻译。
> 原文链接

线性分类

上一节我们介绍了图片分类，这项任务从侯选种类中选一个出来标记一张图片。此外，描述了k-近邻算法(简称：kNN)，它是将需要预测的图片与训练集中的图片一一对比算法。kNN的确定如下：

- kNN分类器必须保存所有训练集，因为在后来的预测中需要对比。这非常耗费空间，因为有时候训练集达到几个G。
- 因为预测新的图片时需要与训练集中的图片对比，所以预测过程非常耗时。

总览

现在要学一个图片分类器，这个分类器要应用后来的神经网络和CNN。这个算法有两个关键：一个得分公式用来计算原数据到不同分类的得分，一个损失公式量化了预测分类的得分与正确分类之间的差距。我们会将这个方式转化为优化问题：调整得分公式参数使用损失公式的值最小。

将图片以参数化的方式映射成类别的得分

第一步是定义一个得分公式，能够将像素值映射成每个分类的得分，我们会用一个实际的例子完成这个过程。如上节，假设训练集为$x_i \in R^D$，每一个样本都配有一个分类标记y_i。这里的$i = 1 \dots N$和$y_i \in 1 \dots K$，意味着有N个训练样本（每个训练样本的维度为D）和有K个不同分类。在CIFAR-10这个训练集中，N=50,000，每个训练样本的维度数是D = 32 x 32 x 3 = 3072 个像素。并且K = 10，这是因为有10个不同分类，如狗、车、猫等等。现在我们开始定义得分函数$f：R^D \mapsto R^K$，它可以使原本的像素映射成不同分类的得分。

线性分类器

本小节会以一个简单的、可论证的线性公式开始：

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

在上面的公式中，x_i表示将一张图片的所有像素变为一个只有一列的向量:D x 1，而矩阵W（大小为[K x D]）和向量b（大小为[k x 1]）是这个公式的参数。在CIFAR-10中，x_i即为一个 3072 *1的列向量，W为10*3072的矩阵，b为10 x 1的列向量。所以用公式计算时，输入的是3072个代表像素的数字，产出却是10个数（代表不同分类）。参数W被称为权重，b因为只影响了公式的得分，但不影响原有的输入数据x_i，而被称为偏差向量。另外，人们经常将参数和权重混合使用。

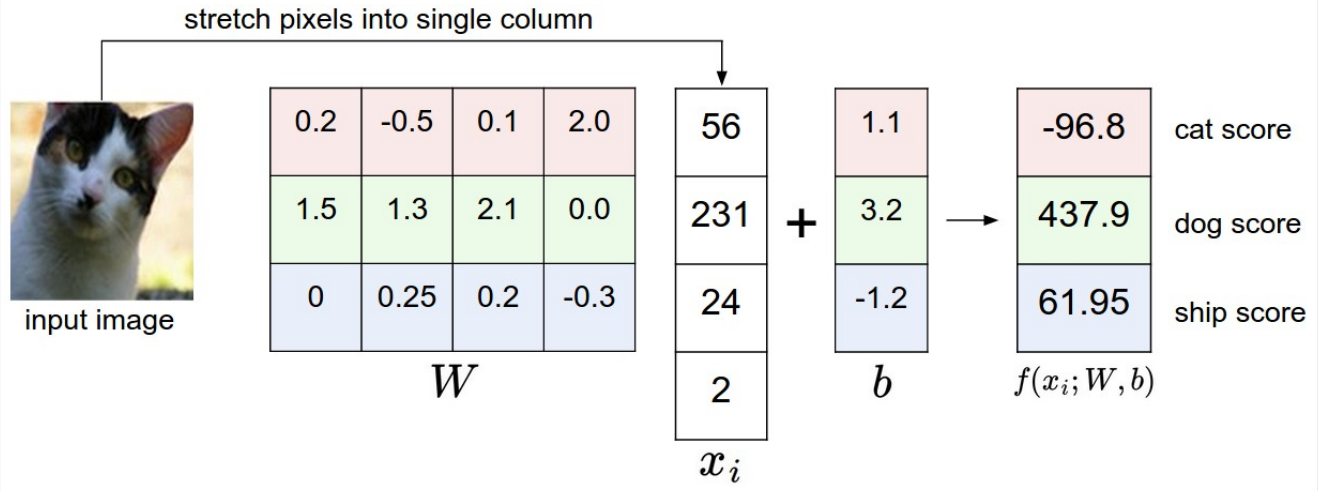
还有些要点如下：

- 注意一次矩阵的乘法Wx_i 实际上分别平行进行了十次分类的评价（每一次对应一个分类）,每一类对应了W中的每一行。
- 注意输入数据(x_i, y_i)是固定不变的，但我们会控制参数W和b的数值。目标是使用训练集，选定W和b的数值，让得分最高的分类等于正确的分类（也就是y_i）。下面会更详细的讨论这个过程。但是目前我们直觉上都希望正确的分类在这个公式中得分最高。
- 还有一个优点是我们只需要利用训练集选择W和b的数值，一定选定，整个训练集都没什么用了，我们只要保留参数的数值就好。这是因为需要预测的新图片只需要通过公式计算得分即可判定分类。
- 注意预测新图片时，只需要进行一个矩阵乘法和加法，这个速度远快于将预测图片和训练集中的图片一一对比。

> 稍稍剧透一下，实际上CNN也是进行了类似上面的过程：将像素直接映射成不同分类的得分，只是CNN的得分公式更加复杂、参数更多而已。

解释线性分类器

线性分类器通过将所有像素（包括三种颜色通道）与权重相乘算出了不同分类的得分。权重的正负、大小决定了特定位置、特定颜色对某一分类的贡献。举例来说，当图片边缘有大量的蓝色（可以对应为水），那么该图片的分类更有可能是蓝色，所以可以预期"船"这个分类器将会把蓝色通道的权重调高，这样出现蓝色会使得船这个分类的得分增加，同时将红、绿色通道的权重设置为负数，也就是说出现红、绿色会降低船这个分类的得分。

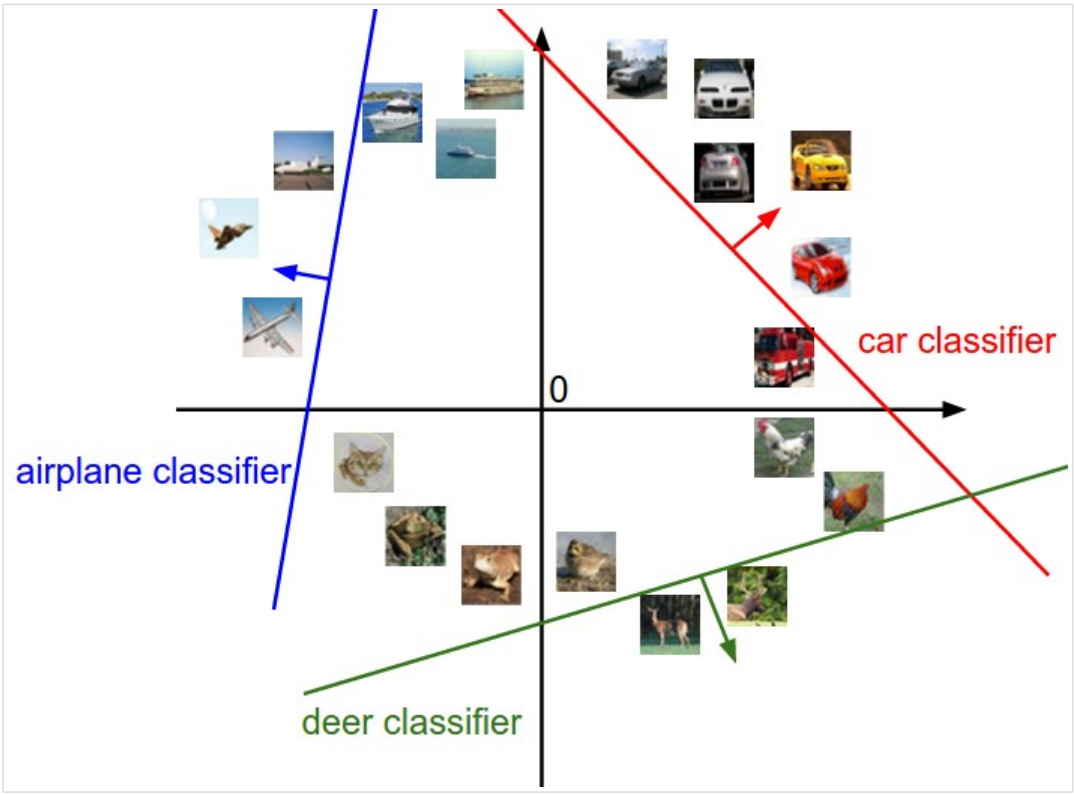


将图片映射成不同分类的得分的例子。为了可视化，假设图片只有4个像素（只有黑白像素，为了简洁不考虑颜色通道），有三个分类：红色代表猫、绿色代表狗、蓝色代表船。（声明一下，这里的颜色仅仅暗示了分类，和RGB颜色通道没有一点关系）。我们将像素表示为列向量并且进行矩阵乘法则得到了每个分类的得分。注意这例子设置的权重不爱合理，因为看起来猫得了低分，狗得了高分，让人们相信觉得分类器会把猫判断成狗。

将图片视作高纬度的点来解释分类器

因为图片能表示成高维度的列向量，我们可以理解每一张图片都作为空间里的一点（比如每一张CIFAR-10的图片都是3072维空间的一个点，3072维是由图片的长宽3，即RGB颜色通道而得来的），同理，整个数据集可以视为一个带有分类标签的点集。

因为定义了每一个种类的得分是权重和像素相乘后求和的结果，所以每一个种类得分都是一个在空间中的线性方程。由于不能可视化3072维的空间，但是我们可以借助两维图像来想像这个过程，分类器做的事如下所示：



上图展现了图片空间，每一个图片是一个点，并且三个分类器被可视化了。以红色为例，红色线上的点表示对于汽车这个类型来说会得到0分。红色剪头表示增长的方向，所有在红色剪头方向的点会得到正分，所有在相反方向的点会得到负分。

> 虽然一个图中的每条线有两种表示方式，比如红线：$x_1 + x_2 - 1 = 0$, 也可以表示为$-x_1 - x_2 + 1 = 0$, 此时如果带$x_1 = 2, x_2 = 2$进入第一个公式则为正数、第二个公式却为负数，不符合在图中剪头的方向的点得分越高。实际上，第一个公式和第二个公式完全是两码事，因为有着不同的W、b，划线的时候虽然可以等于0，但是公式中等号另一边是函数值：$f(x_i, W, b) = Wx_i + b$, 所以不能随意调换。

如上所述，W中的每一行都是一个分类器。通过图形的观察，可以知道当我们改变某一行的 W 时，对应得改变图形中线条的方向（转动线条）。而偏差b，则是允许我们移动线条。如果没有偏差，当$x_i = 0$时，只能得到0，所以所有的线条都会穿过原点。

将分类器视作模板匹配

另一种 W 的解释是将每一行其视作一个类别的模板（或者原型），每一个类型的得分就是将图片与模板用内积（也称点积）的匹配程度。在这种情况下，分类器在做模板匹配，模板是提前学习好的。另一种角度就是我们仍然在做最近邻算法，只是预测图片不是和训练集中的每一张图片做比较，而是和模板做比较（模板又是通过训练集训练得来的，它不必是训练集中的任意一张图片），然后我们使用（负）点积作为距离，而不是L1、L2距离。

上面显示的图片表示图片已经通过训练集学习好了，比如船的模板包含了大量的蓝色，待预测的图片如果包含大量的海洋则会得到较高的分（点积计算）。

> 很好奇这个图片是怎么得到的。

此外，观察马的模板感觉里面好像有两个马头，这是因为训练集中有两种朝向的马头。线性分类器把不同方向马头的图片都融合成了一张。同样，汽车的模板貌似也融合多了个朝向和颜色的汽车，模板中汽车呈现出红色，这说明在CIFAR-10的训练集中红色汽车比较多。线性分类器处理多个颜色汽车的能力较差，但是之后学习的神经网络能极好的完成这个任务。稍微透露下之后的内容，神经网络将会开发出多个中间的神经元作为隐藏层，这样就可以识别多种类型的汽车了（比如朝左边的绿色车，朝前的蓝色车），同时下一层的神经元能够将多个汽车探测器的结果组合起来得到一个更准确的汽车类别的得分。

偏置的作用

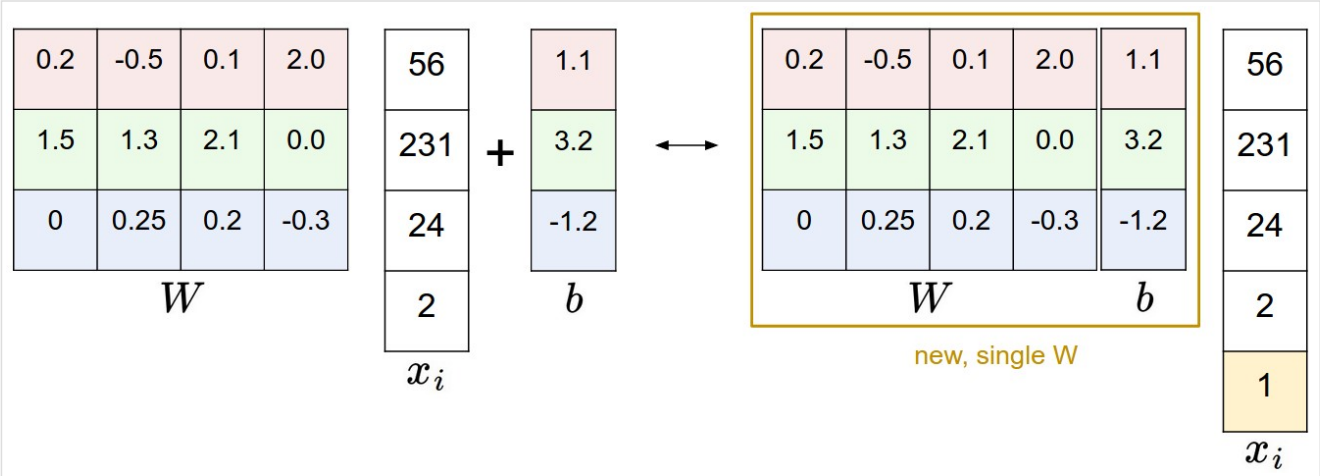
下面我们将学习一个技巧将两个参数W和b合并成一个。首先回顾一下得分公式的定义：

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

分别处理两个参数略显繁琐，一个常用技巧就是将W和b组合成一个新的矩阵，同时对向量x_i增加额外的一个维度，并且设置常亮为1，这个维度称为偏置维度。有了偏置维度，得分公式的简化如下，这样可以只进行一次矩阵乘法。

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

因此，在CIFAR-10的例子中，x_i的是[3073 x 1]，而不是[3072 x 1]（额外的一个维度是常量1），同时W是[10 x 3073]，而不是[10 x 3072]，额外的一列对应偏置。下图展示了这些变化：



阐明偏置技巧的过程。原来是先做一次矩阵乘法，再加上偏置。这样的做法等于把偏置在W的最后一列，并对任何输入向量多加一行常量1。因此，如果我们预处理了数据，对每一个输入最后都加一行常量1的话，我们只需要学习一个矩阵（包含权重和偏置）。

图片的预处理

为了快速授课，之前一直使用的是原本的像素数值（范围是从[0,255]）。但在机器学习的过程中，常见做法是归一化我们输入的数据（因为是图片，恰好可以认为已经归一化了，因为都是像素，同一个特征）。实践中，中心化数据也非常重要，中心化可以通过让每个数值减去平均值完成。以图片为例子，这个过程是计算出训练集的图片中像素的平均值，再让每个像素减去均值，这会使得处理后数据的范围大约在[-127,127]。更进一步的预处理是放大缩小使得每一个特征的范围都在[-1,1]之间。在我们学习动态梯度下降后，就会明白均值为0有多么重要。

损失函数

上一节我们定义了一个公式可以将像素映射成类别的得分，这是乘以权重后的结果。此外，我们也不能改变训练集的数据(x_i, y_i)，但是我们可以改变参数，并且使得预测得到最高分的分类尽可能的是正确的分类。

举例说来，回顾上面的例子，一只猫的图片被预测了类别为猫、狗、窗的得分，可以看到那个例子中的参数不是很好，预测类别为猫的得分是-96.8，远低于预测为狗的437.9和预测为船的61.95。我们需要一种办法来测量这样不正确，那就是使用损失函数（又被称为代价函数或者目标）。直觉上，我们希望预测错误损失越大，预测正确损失越小。

多类别支持向量机的损失

很多方式可以定义损失函数。作为第一个例子，我们会开发一个常用的损失函数：被称为多类别支持向量机。设置SVM损失的关键是让正确分类的得分比错误分类的得分高出一个Δ。有时候拟人化损失函数有助于我们理解：SVM想要一个特定参数组合，这个参数组合产生最低的损失。

更详细地说，对于x_i和其类别y_i（只是正确类别的索引）。得分函数算出各个分类的得分$f(x_i, W)$，被简写为s，那么第j类的得分就是第j个元素：$s_j = f(x_i, W)_j$。那么对于x_i而言，多类别SVM的损失公式如下：

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

例

用一个例子解释这个公式，假设现在有三个得分为$s = [13, -7, 11]$，并且第一个分类为正确分类（$y_i = 0$）。假设Δ是10，这个超参数我们待会再详细讨论。上面这个公式需要将所有不正确的分类相加，所以在本列中只有两项。

$$L_i = \max(0, -7 - 13 + 10) + \max(0, 11 - 13 + 10)$$

可以看到第一项等于0，因为[-7-13+10]得到一个负数，使用$max(0, -)$作为一个阈值函数使得只能得到0。于是，因为正确得分13大于错误分类的得分-7超过了10，所以对于这一对儿来说我们得到的损失为0。实际上，差值为20，这个远大于10，但是SVM只关心在小于10的差值，差值大于10的情况下会被max函数忽略掉。第二项计算[11-13+10]得到8，尽管正确的分类比错误的分类大，但是没有超过理想的阈值10，其实只有22点，所以经过max函数会得到8点损失（这个损失的含义也是还差多少就达到理想的差值）。总之，SVM损失希望正确分类的得分比错误分类的得分大于一个阈值，用Δ表示，如果没有大于这个阈值，那么将累计损失。

因为在本节中，我们主要使用线性的得分函数:$f(x_i; W) = Wx_i$，所以重写损失函数：

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

其中w_j是W的第j行被变形为一列。其实，当我们考虑更为复杂的得分函数时，就不会使用这种形式了。

最后有一个术语需要了解，就是在$(max(0, -)$的中设置为0的时候，这个公式常被称为hinge loss。另外，平方的hinge loss SVM（被称为L2-SVM）也经常被使用，其形式为：$(max(0, -))^2$，它以平方而不是线性的形式加重惩罚。未平方的版本是一个标准版本，但是在某些数据中平方后效果会更好。具体使用哪个可以通过交叉验证来确定。

> 损失函数度量了我们对预测结果不满意的程度

多类别支持向量机想要正确分类的得分大于别的分类的得分至少超过一个delta。如果有错误分类的得分在红色范围内（甚至更高），那么将会积累损失，否则损失为0。我们的目标是找到权重参数对于所有的训练集样本都满足这个条件，同时尽可能的降低损失

正则化

刚说的损失公式有一个bug。假设不止一个W能够正确分类所有样本（也就是说有所有样本的得分都满足了大于错误分类一定差异，对所有i来说：$L_i = 0$）。换句话说，W并不是唯一的，有几个相似的W能够正确分类所有样本。一种简单的办法可以找到这一些相似的W，如果某一个W能够正确分类所有样本，那么这个W的倍数，也就是λW，其中$\lambda > 1$也能正确分类所有样本呢，也能得到0的损失。出现这种情况的原因是是这种转变只是同时增大了所有分类的得分和差距。比如说，正确分类的得分和得分最高的错误分类的得分的差距是15，那么当W翻倍时，新的差距也就是30。

换句话说，我们需要一种办法在一群类似的W中提高某一个W的被选中的优先级，从而消除这样的歧义。为损失函数加一个正则化的惩罚的参数：$R(W)$ 即可。最常用的正则化惩罚参数就是L2距离，它会降低数值数值较大的W被选中的可能性，也就是说提高了小数值的W被选中的可能性。

$$R(W) = \sum_k \sum_l W_{k,l}^2$$

注意看上面的图，W是一个矩阵。在上面这个公式中，我们把W矩阵中的元素平方后全都加起来了。注意这个正则化公式并不是对数据的操作，而仅仅是对权重的操作。多分类SVM加上正则权重惩罚项后才得以完善，因此多分类SVM由两部分组成：数据的损失（它是指平均了L_i所有样本的损失）和正则化损失。完整的公式如下：

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

将上面的公式展开：

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

其中N是样本的个数。正如你所见到的一样，我们给正则化的惩罚参数加了一个表示权重的超参数λ。通常除了交叉验证，没有什么简单的办法能够确定λ。

除了上面描述的动机外，我们还有很多别的原因为损失函数增加正则化项，其中大多数原因在以后的章节中都会提及。比如说，已经证明将L2距离作为正则化项还可以帮助找到SVM中的最大间距（max margin），如果有兴趣请查阅CS229获得更多内容。

当然最大的好处就是能够惩罚数值较大的W从而选择了数值较小的W，这有助于改进泛化，实际上，我们希望W的维度尽可能多，如果少的话将会对得分有着极大的影响。举个例子来说明这一点，假设，有一个输入向量 $x = [1, 1, 1, 1]$，还有两个权重向量：$w_1 = [1, 0, 0, 0]$，$w_2 = [0.25, 0, 25, 0.25, 0.25]$。因为$w_1^T x = w_2^T = 1$，两者点积相同，然而$w_1$的L2惩罚是1，但是w2的L2惩罚却只有0.25。因此，可以知道w_2将会优先被选择。直观上来看，w_2的数值更低，而且每一个维度都有数，显得更加均匀。正是因为L2惩罚倾向于小数值和更均匀的权重，才使得最终的分类器倾向于将所有维度的数据都考虑进去，而不是单单只考虑几个权重太高的维度（太奇妙和精彩了）。之后我们将看到，这通过降低了过度拟合从而改进了泛化能力。

注意偏置(bias)并不需要这样的效果，因为偏置不像权重，偏置并不会控制某个维度的数据的对得分影响的程度，反正偏置最后也就加一下而已，某个维度的数据并没有加强。因此，通常只正则化参数W，不正则化偏置b。然而，实践中，就算正则化了b也不会带来什么影响。最后，注意因为正则化的惩罚项，我们永远不可能不得到损失（不存在对某个样本数据损失为0），显然这是因为不太可能出现W=0的情况（W是矩阵，W=0意味着所有矩阵元素都为0）。

代码

下面是没有实现正则化项的损失函数的跑python代码，分为非向量化的版本和半向量化的版本：

```python
# -*- coding: utf-8 -*-
import numpy as np

def L_i(x, y, W):
  """
  未向量化的版本：算出一个样本(x,y)多分类svm的损失
  - x是代表一个图片的列向量（比如在CIFAR-10就是 3073 x 1 ）
   注意这是将偏置放在了第3073行
  - y 是这个图片正确的分类的索引（比如在CIFAR-10 0到9之间的数字）
  - W 就是权重矩阵(比如在CIFAR-10中就是10 x 3073)
  """
  delta = 1.0
  scores = W.dot(x) # 对每个分类的得分，注意得到的是 10 x 1 的矩阵
  correct_class_score = scores[y]
  D = W.shape[0] # W.shape 会得到 10 * 3073，第一个元素就是10
  loss_i = 0.0
  for j in xrange(D): #注意只对错误的分类循环
    if j == y:
      # 跳过正确分类
      continue
    # 积累第j个分类的损失
    loss_i += max(0, scores[j] - correct_class_score + delta)
  return loss_i

def L_i_vectorized(x, y, W):
  """
  一个速度更快的半向量化版本，注意对于一个样本来说这个版本并不需要for循环。
  但是仍然在函数外需要for循环来遍历所有的样本点。
```

```
    """
    delta = 1.0
    scores = W.dot(x)
    # 用了一个向量操作，我看就是numpy的一个api，来算出来所有分类的差距
    margins = np.maximum(0, scores - scores[y] + delta)
    # 因为第y个位置是正确的分类，所以希望不记入损失中去。所以我们将第y个位置的损失置为0
    # 我们只考虑错误分类的损失
    margins[y] = 0
    loss_i = np.sum(margins)
    return loss_i

def L(X, y, W):
    """
    #全向量版本的实现
    - X 全部的训练集，一列就是一个样本(e.g. 在CIFAR-10中：3073 x 50,000 )
    - y 一个标明正确分类的数组 (e.g. 50,000元素的数组)
    - W 权重(e.g. 在CIFAR-10中：10 x 3073)
    """
    # 别用循环，算出所有训练集样本的损失
    # 哈哈，这是作业~~
    delta = 1.0
    total_scores = W.dot(X)
    #做一个和total_scores一样的矩阵，然后每一列都是正确分类的得分
    #这怎么可能不用循环，不用循环岂不是累疯了，取出每一个样本正确的分数
    right_scores = [total_scores[y[i],i] for i in xrange(X.shape[1])]

    r_s = np.tile(right_scores,(total_scores.shape[0],1))
    margins = np.maximum(0, total_scores - r_s + delta)

    #同样把正确分类的距离至为0
    for i in xrange(margins.shape[1]):
        margins[y[i],i] = 0

    return np.sum(margins)


def main():
    ##模拟数据
    X = np.random.randint(255,size=(3073,50000))
    y = np.random.randint(9, size=50000)
    W = np.random.rand(10,3073)
    #测试L_i和L_i_vectorized能得到相同的结果否
    L_i_loss = 0
    L_i_vectorized_loss = 0
    for column in xrange(X.shape[1]):
        x = X[:,column]
        one_y = y[column]
        L_i_loss += L_i(x,one_y,W)
        L_i_vectorized_loss += L_i_vectorized(x,one_y,W)
    L_loss = L(X,y,W)
    print('L_i_loss:%s'%(L_i_loss))
    print('L_i_vectorized_loss:%s'%(L_i_vectorized_loss))
    print('L_loss:%s'%(L(X,y,W)))

if __name__ == '__main__':
    main()
```

还有一点记住，SVM损失度量了分类器预测训练集中的分类与正是分类相同的程度。另外，损失值越低，对训练集的预测越好（与正确分类相同）。

> 现在我们必须做的事情就是找出能得到最小损失的权重。

实践时的建议

Δ的大小

注意我们忽略了对如何设置Δ大小的讨论。到底大小应该设置成多少？还是我们必须用交叉验证来验证它吗？实践证明在所有的应用中都可以将Δ设置为1。超参数Δ和λ看起来像两个参数，实际上他们都在做同样的权衡：数据损失和正则化损失。理解这个的关键就是W的大小对得分有着决定性的影响（当然对分数之间的差距也有着影响）。当我们将W中所有的值都压低时，得分自然会低；当我们拉高W的值时，得分自然就高。因此，得分之间的差距Δ到底设置成多少才合理呢？是$\Delta = 1$还是$\Delta = 100$几乎没了任何影响，反正W也可以随意的设置值，从而引起分数的差异。所以，真正能够决定正则化损失的占比（权重）的参数实际是是λ。

与两分类SVM的关系

也许有在此之前有些机器学习的知识，比如比较了解两分类SVM，其中第i个样本的损失可以写作：

$$L_i = C \max(0, 1 - y_i w^T x_i) + R(W)$$

其中C是一个超参数，并且$y_i \in -1, 1$。可以认为两分类SVM是本节上面讲得多分类SVM的公式的一种特殊情况，也就是当分类只有两种的时候，那么损失函数就如同上面这个公式。此外，在上面这个公式中的C也平衡着数据损失和正则化损失，与λ的关系是：$C \propto \frac{1}{\lambda}$。

优化最初的形态

如果有已经了解过SVMs，你肯定也听过核技法、双重算法、SMO算法等等。因为本课主要焦距在神经网络，所以我们的优化对象总是最初的方程式。这些对象其实技术并不是每一个都可微分（比如max(x,y)方式就不是，因为当x=y时就无法区分了），但是实践中这并不是什么问题，我们通常可以使用次梯度法。

其他多分类SVM的公式

主要本节所使用的多分类SVM只是表达多分类SVM众多公式之一。另一种常见的形式是 One VS ALL (OVA) SVM，它会训练一个个单独的分类器去分类每一个单独的分类和剩下的分类，也就是一个分类 vs 剩下的分类。很少在实践中看到ALL VS ALL的策略。我们的公式是遵从了Weston and Watkins 1999 (pdf)的版本，它提供了一个比OVA更优秀的方式（你可以创建一个多分类数据集，然后得到用论文的版本实现数据损失为0，但是OVA不行，如果感兴趣请阅读论文）。最后一个介绍的公式是结构化的SVM，它会最大化正确分类的得分与得分最高的错误分类的差距。清楚这些公式的区别已经超出了本书的范围。反正在本节中提到的公式足够我们在实践中使用了，不过有人也证明了最简单的OVA公式也得得到很好的结果（请见由Rikin的论文：In Defense of One-Vs-All Classification (pdf)）

Softmax分类器

SVM只是常用的分类器之一，另一个则是Softmax分类器，它有着不同的损失函数。如果你之前听说过逻辑回归分类器，那么Softmax分类器则逻辑回归分类器的在多分类上的泛化。不像SVM对产出的解释：每个分类的得分，这个得分没有统一的标准（不同分类器的得分不是一个标准），而且又难以解释。Softmax给出了符合直觉的解释（归一化后表示某分类的概率），并且有着概率的解释（下面会简短介绍）。在Softmax分类器中，映射公式：$f(x_i; W) = Wx_i$并没有发生变化，但我们会把这些分数解释为每个分类未归一化的log概率，并且用交叉熵损失来替换hinge损失，交叉熵的公式如下：

$$L_i = -\log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}\right) \qquad \text{or equivalently} \qquad L_i = -f_{y_i} + \log\sum_j e^{f_j}$$

其中，我们用f_j来表示类别得分向量f中的第j个元素。像原来一样，对数据集全部的损失就是所有训练集样本的损失L_i的均值，再加上一个正则化项：$R(W)$。
其中括号内的就是Softmax公式：

$$f_j(z) = \frac{e^{z}j}{\sum_k e^{z}k}$$

它接受一个任意分类得分的向量，然后把它们都塞进另一个元素的值都在0到1之间，并且所有元素的值加起来为1的向量中去。

> 作为入门，作者在这里确实介绍的非常简洁。实际上，Softmax公式里不用e的次方，也可以达到每个元素的值都在0到1之间，并且所有元素的值加起来为1这个效果，其实这就是简单的占比。比如说三个元素$[50, 25, 25]$，只看占比的话很快就能算出来是无非就是$[50/100, 25/100, 25/100] = [0.5, 0.25, 0.25]$，但是Softmax的结果呢？

```
In [52]: np.exp(50)+np.exp(25)+np.exp(25)
Out[52]: 5.1847055287310814e+21

In [53]: _sum=np.exp(50)+np.exp(25)+np.exp(25)

In [54]: np.exp(50)/_sum
Out[54]: 0.99999999997222422

In [55]: np.exp(25)/_sum
Out[55]: 1.3887943864578274e-11

In [56]: np.exp(50)/_sum + np.exp(25)/_sum + np.exp(25)/_sum
Out[56]: 1.0

```

> 可以看到，是[0.99999,0.0000001,0.00001]，所以可以看出，借由e次方函数的特点，Softmax极大的拉高了得分较高的分类的占比（概率）。

```
In [57]: _sum=np.exp(50)+np.exp(51)

In [58]: np.exp(50)/_sum
Out[58]: 0.2689414213699951

In [59]: np.exp(51)/_sum
Out[59]: 0.7310585786300049
```

> 即使在得分仅仅相差1的情况下，差距也非常大。
> 再仔细观察交叉熵损失公式，取的是-log函数，此函数当x的值接近于1时，y值为0；当x值接近于0时，y值无限增大；可以看到交叉熵损失公式的分子实际上就是正确分类的得分，得分越高，损失越低。

全部损失的交叉熵公式第一次看的话看起来有点吓人，但其实它非常好解释。

从信息论的观点来解释

交叉熵在正确分类的分布和预测分类分布之间的定义如下：

$$H(p,q) = -\sum_x p(x)\log q(x)$$

因此Softmax分类器实际上是在缩小预测分类概率(正如上面的$q = e^{f_{y_i}} / \sum_j e^{f_j}$)和正确分类分布之间的差距，所谓正确分布可以理解成一个在正确分类上的概率质量函数($p = [0, \dots 1, \dots, 0]$，它只有在x轴上$j_i$处的一个为1的值。此外，交叉熵也可以被写成一个和Kullback-Leibler距离相加的形式：$H(p,q) = H(p) + D_{KL}(p||q)$，因为我们视正确分布为稳态，p又代表正确的分布，所以第一项$H(p)$为0。那么现在就像等于在缩小两个分布之间的KL距离。换句话说，交叉熵的最终目标是想要预测的分布和由正确分类形成的分布一模一样。

> 读完了也没看出交叉熵损失公式和交叉熵有什么关系？看起来交叉熵损失首先用Softmax公式极大的拉高的得分高的分类的占比，其次就是使用了一下-log函数，算损失而已。

从概率论的观念来解释

公式：

$$P(y_i \mid x_i; W) = \frac{e^{f_{y_i}}}{\sum_j e^{f_j}}$$

可以解释为在给定图片x_i和参数W的情况下，被预测到正确分类y_i的概率。为了得到这个解释，Softmax分类器是将f向量中的得分视作未归一化的log概率，再对这些值取幂（就消除了以e为底的log），就获得了未归一化的概率，最后作除法进行归一化，让所有的概率相加为一。在概率论的观点里，因为损失方程表示损失，为了减小损失，我们可以理解方程其实是在最小化正确分类的负的log的可能性，也可以认为是在进行MLE（最大化正确分类的概率的可能性，Maximum Likelihood Estimation）。这个观点有一个特点，就是可以认为全部训练集的总损失中的正则化项$R(W)$是一个高斯分布（我觉得就是先验概率的参数），而不是一个权重矩阵，因此我们不是在进行MLE，而是再进行最大化后验（MAP， Maximum posteriori estimation）。这里提到这些只是为了加强大家对公式的认识，更多细节超出了本课的范围。

实践问题：数值稳定

当编码完成成Softmax的时候，有些项比如$e^{f_{y_i}}$ 和$\sum_j e^{f_j}$取幂后数值非常大。除以大数可能会造成数据的不稳定，所以使用归一化技巧是非常重要的。如果我们在分子分母上乘上一个常数C，当然分母求和公式中所有项都会乘上一个C，公式如下：

$$\frac{e^{f_{y_i}}}{\sum_j e^{f_j}} = \frac{Ce^{f_{y_i}}}{C\sum_j e^{f_j}} = \frac{e^{f_{y_i}+\log C}}{\sum_j e^{f_j+\log C}}$$

C的取值可以随意，这不会改变结果，但是我们使用这个数改进了计算的稳定性。通常我们会选择C为$logC = -\max_j f_j$。这只是简单将向量内所有每个元素都减去了向量内的最大值而已。代码如下：

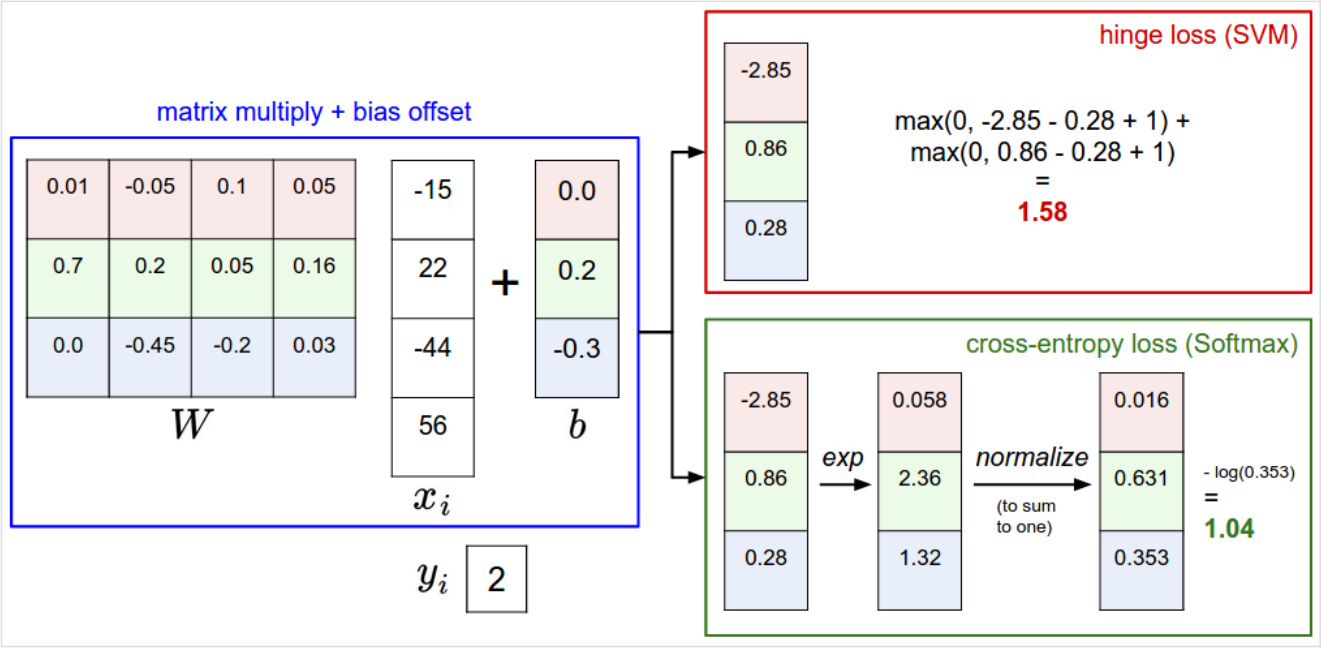
```
f = np.array([123, 456, 789]) # 三个分类，并且有一个分类的分数特别高
p = np.exp(f) / np.sum(np.exp(f)) # 取幂后数值非常大，造成溢出
#p的输出：array([  0.,   0.,  nan])

# 让所有的元素减去最大值，这样元素中最高的
f -= np.max(f)
#f输出：[-666, -333, 0]

p = np.exp(f) / np.sum(np.exp(f)) # 没问题，可以拿到正确的值
#p的输出：array([  5.75274406e-290,   2.39848787e-145,   1.00000000e+000])
```

对命名的困惑

准确地说，SVM分类器使用hinge loss，偶尔也会被称为最大间距损失(max-margin loss)。Softmax分类器实际上使用交叉熵损失。Softmax分类器的命名起源于Softmax公式，该公式也只是将每个类的分数换算成另一个正数，并确保换算后所有的数加起来等于1，这样才能算出交叉熵。所以，并没有Softmax损失这样的叫法，Softmax只是一个压缩（换算的）方程，但是通常会在速记时写成Softmax损失。

SVM 与 Softmax

下图让帮你理清这些关系：



SVM分类器和Softmax分类器对于一个样本点的区别。对两个分类器，我们都算出了同样的得分向量（通过矩阵乘法），区别在于对分数向量的解释，SVM通过计算正确分类的分数（下标为2的分类，第三个分类，用蓝色表示）是否高于别的分类分数一个差距来得损失。然而Softmax分类器将每一个分数解释成未归一化的log的概率，并且希望归一化后正确分类的概率的log尽可能的高（也就是它的负数尽可能的低）。最终的损失，对于SVM是1.58，而对于Softmax是1.04，注意这两个数不可比较，他们与在相同数据集和相同分类器下产出的数值比较才有意义。

Softmax分类器提供了每个分类的概率

不像SVM计算出来的数没有一个标准来恒定和解释，Softmax分类器则使我们计算出来所有分类的概率。比如，SVM分类器给你的结果是一堆分数[12.5,0.6,-23.0]，对应着猫、狗、船。但是Softmax分类器将会为着三个分类给一个概率[0.9,0.09,0.01]，这使我们可以非常清楚属于某个分类的程度。我之所会对概率打上引号，是因为各个分类概率的大小依赖于正则化项λ，它其实是作为分类器的输入，它控制着概率分布是否平坦。举例来说，三个分类得分为[1,-2,0]，使用softmax公式计算出:

$$[1, -2, 0] \rightarrow [e^1, e^{-2}, e^0] = [2.71, 0.14, 1] \rightarrow [0.7, 0.04, 0.26]$$

这一步算以e为底的得分的幂，然后归一化。现在我们把λ项设置的更高，这样的话W会得到比较重的惩罚，以至于W的数会变小。当输入样本不变的情况下，W变小，最终的得分自然而然也会变小，假设最终的得分只有原来的一半，也就是[0.5, -1, 0]。那么用Softmax公式计算出：

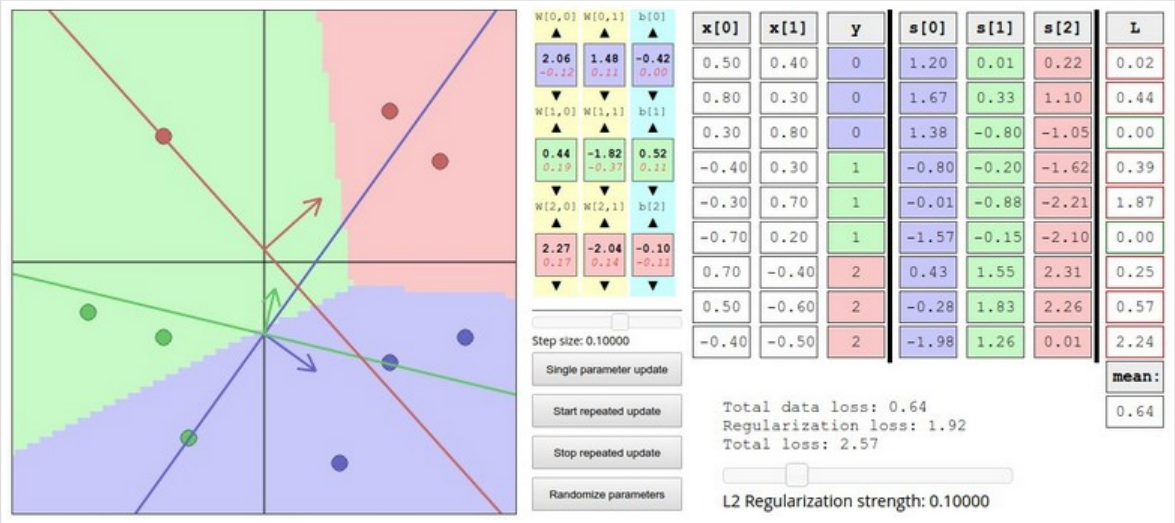
$$[0.5, -1, 0] \rightarrow [e^{0.5}, e^{-1}, e^0] = [1.65, 0.37, 1] \rightarrow [0.55, 0.12, 0.33]$$

可以清楚得看到，概率分布的更为平滑了。所以，当设置正则化项λ的数过大时，权重W将会倾向于产出较小的数，这时最终产生的概率分布比较平坦。另外，由Softmax分类器计算出的概率确实可以表明属于某个分类的程度，但是对于SVM的产出，得分和得分之间的差距都不能说明什么问题，唯一还有点用的就是由得分高低排出不同分类的顺序。

实践时，SVM和Softmax性能相当

其实，SVM和Softmax两者间的性能非常小，不同的人对哪个更好有着不同的观点。与Softmax分类器相比，SVM很容易陷入局部最优(local objective,这个词最后从后面的例子来看懂是什么意思，反正就是优化到了一定程序，SVM很容就不再优化了)，这可以认为是一个bug或者是特点。举例子来说，得分是[10,-2,3]，其中第一个是正确的分类，一个SVM分类器（假设设置理想中的差距$\Delta = 1$）只会得到0的损失，这是因为第一个分类的得分比错误分类的得分大得已经超过了Δ，就不会积累损失。SVM不会在乎具体的分数是多少：无论是[10,-100,-100]还是[10,9,9]，反正错误分类与正确分类之间的差距都大于等于1，所以损失为0。但是同样的场景对于Softmax却有着完全不同的表现，[10,9,9]会比[10,-100,-100]得到高得多的损失。所以当得到上述其中之一的结果，SVM分类器是不会再进行优化了，因为损失都0了，还怎么优化？但是Softmax不同，Softmax永远不会满足于得分：因为正确分类的概率可以越来越高，错误分类的概率可以越来越低，损失总是可以越来越小。然而SVM产出的正确分类和错误分类之间的差距一旦满足了，它不会做更多的优化，它也不会再在乎具体分数是多少。我们可以明显的感受到这个特点：举例子来说，一个汽车分类器核心问题是不受到某个特别分类的影响，比如说雾，当雾已经有很低的得分了，而且雾也聚集在了远离汽车的另一端，但这个汽车分类器还是费了很多资源将区分卡车和汽车。很显然，Softmax很容易陷入这种的情况，但是SVM分类器却能避免这样的情况，因为卡车的得分与汽车的得分保持可以忍受的差距就行了，而不像softmax是无穷尽的优化。

可交互的web样例



这个web样例可以帮助我们直观地感受感受线性分类器。这个web样例使用3个类和二维的数据，视觉化了本章讨论的损失函数。这个例子也会进行优化，这是我们下一章要讨论的内容。

总结

- 我们设计了一个得分公式，能够将像素值映射成每个分类的得分（本节中，就是一个依赖权重和偏置的线性公式）。
- 不像kNN分类器，参数化方式的优势使得我们一旦通过训练集获得了参数，我们就可以抛弃训练集。此外，预测一张图片的数据非常快，因为只需要让像素和W进行一次矩阵的乘法，而不用和训练集中的图片一张一张的对比。
- 我们介绍了偏置技巧，它允许我们将偏置加入到权重的矩阵中，这样我们只关心权重矩阵就可以了。
- 我们定义了损失函数（并介绍了两种对线性分类器常用的损失计算方式：SVM和Softmax），它可以度量一组参数预测的训练集中的分类和正确分类的差距。定义的损失函数当预测正确时，只会到很小的损失。

现在我们可以把训练集的图片依据一堆参数映射成不同分类的得分，我们也使用两个损失函数，这可以用于评价我们的在训练集上的预测。但目前的问题是我们如何决定这堆参数，让其产出最小的损失。这个过程称为优化，下一节将详细介绍。

更多阅读

可选的，也是有很多有趣的观点。

- Deep Learning using Linear Support Vector Machines：2013年Charlie Tang陈述观点：L2 SVM是比Softmax更优秀。

#cs231n

❮【译】cs231n第一课:图片分类、数据驱动模型，k近邻算法，训练集、验证集、测试集的划分

【译】cs231n第三课：优化 - 随机梯度下降❯

zhangyang's blog正在使用多说