loss function

SVM损失函数

“碗”型结构，所以为凸函数

由于SVM中的max操作，导致其在部分点不可微（引入次梯度）

优化

核心思想：迭代优化。我们的方法将是从一个随机的 W 开始，然后迭代地优化它，每次都使其略微更好。

沿梯度方向优化

用有限差分法计算梯度

```
def eval_numerical_gradient(f, x):
 """
 a naive implementation of numerical gradient of f at x
 - f should be a function that takes a single argument
 - x is the point (numpy array) to evaluate the gradient at
 """

 fx = f(x) # evaluate function value at original point
 grad = np.zeros(x.shape)
 h = 0.00001

 # iterate over all indexes in x
 it = np.nditer(x, flags=['multi_index'], op_flags=['readwrite'])
 while not it.finished:

   # evaluate function at x+h
   ix = it.multi_index
   old_value = x[ix]
   x[ix] = old_value + h # increment by h
   fxh = f(x) # evalute f(x + h)
   x[ix] = old_value # restore to previous value (very important!)

   # compute the partial derivative
   grad[ix] = (fxh - fx) / h # the slope
   it.iternext() # step to next dimension

 return grad
```

可以用来计算任何点，任何函数的梯度：

```
# to use the generic code above we want a function that takes a single argument
# (the weights in our case) so we close over X_train and Y_train
def CIFAR10_loss_fun(W):
 return L(X_train, Y_train, W)

W = np.random.rand(10, 3073) * 0.001 # random weight vector
df = eval_numerical_gradient(CIFAR10_loss_fun, W) # get the gradient
```

用来更新W：

```
loss_original = CIFAR10_loss_fun(W) # the original loss
print 'original loss: %f' % (loss_original, )

# lets see the effect of multiple step sizes
for step_size_log in [-10, -9, -8, -7, -6, -5,-4,-3,-2,-1]:
 step_size = 10 ** step_size_log
 W_new = W - step_size * df # new position in the weight space
 loss_new = CIFAR10_loss_fun(W_new)
 print 'for step size %f new loss: %f' % (step_size, loss_new)

# prints:
# original loss: 2.200718
# for step size 1.000000e-10 new loss: 2.200652
# for step size 1.000000e-09 new loss: 2.200057
# for step size 1.000000e-08 new loss: 2.194116
# for step size 1.000000e-07 new loss: 2.135493
# for step size 1.000000e-06 new loss: 1.647802
# for step size 1.000000e-05 new loss: 2.844355
# for step size 1.000000e-04 new loss: 25.558142
# for step size 1.000000e-03 new loss: 254.086573
# for step size 1.000000e-02 new loss: 2539.370888
# for step size 1.000000e-01 new loss: 25392.214036
```

- 该方法计算量较大，因为要对每一个 参数都要进行一次计算
- 步长的选择需要合理，如果步长过大会过头
- 这个方法通常用来检查

用微积分解析计算梯度

利用反向传播技术计算各个变量得到梯度

1. 先画出损失函数的计算图
2. 从后往前计算出每一个节点输出关于输入的梯度（本地梯度）
3. 利用链式法则即可计算出。

- 当一个节点连接多个节点时，梯度在这个节点进行累加

随机梯度下降

实际应用中因为样本量过大，通常会采取随机采小样本数量计算损失函数，来更新W