W:(3073,10)

X\_dev: (500,3073)

y\_dev: (500,1)

def svm\_loss\_naive(W, X, y, reg):

    """

    Structured SVM loss function, naive implementation (with loops).

    Inputs have dimension D, there are C classes, and we operate on minibatches

    of N examples.

    Inputs:

    - W: A numpy array of shape (D, C) containing weights.

    - X: A numpy array of shape (N, D) containing a minibatch of data.

    - y: A numpy array of shape (N,) containing training labels; y[i] = c means

      that X[i] has label c, where 0 <= c < C.

    - reg: (float) regularization strength

    Returns a tuple of:

    - loss as single float

    - gradient with respect to weights W; an array of same shape as W

    """

    dW = np.zeros(W.shape) # initialize the gradient as zero

    # compute the loss and the gradient

    num\_classes = W.shape[1]# 10

    num\_train = X.shape[0]# 500

    loss = 0.0

    for i in range(num\_train):

        scores = X[i].dot(W) #1x10

        correct\_class\_score = scores[y[i]]

        for j in range(num\_classes):

            if j == y[i]:

                continue

            margin = scores[j] - correct\_class\_score + 1 # note delta = 1

            if margin > 0:

                loss += margin

    # Right now the loss is a sum over all training examples, but we want it

    # to be an average instead so we divide by num\_train.

    loss /= num\_train

    # Add regularization to the loss.

    loss += reg \* np.sum(W \* W)

    #############################################################################

    # TODO:                                                                     #

    # Compute the gradient of the loss function and store it dW.                #

    # Rather than first computing the loss and then computing the derivative,   #

    # it may be simpler to compute the derivative at the same time that the     #

    # loss is being computed. As a result you may need to modify some of the    #

    # code above to compute the gradient.                                       #

    #############################################################################

    # \*\*\*\*\*START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)\*\*\*\*\*

    for i in range(num\_train):

        scores = X[i].dot(W) #1x10

        index = y[i]  # 正确的索引

        for j in range(num\_classes):

            if j != index and scores[j] >= scores[index]-1:

                dW[:,j] += X[i]#数据分类错误时的梯度

                dW[:,index] -= X[i]#数据分类正确时的梯度，所有非正确的累减

    dW /= num\_train

    dW += 2\*reg\*W

    # \*\*\*\*\*END OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)\*\*\*\*\*

    return loss, dW