计算机科学与技术学院神经网络与深度学习课程实验报告

实验题目：图像识别与分类		学号：201900130151
日期：2021.10.2	班级： 人工智能	姓名： 莫甫龙
Email：m1533979510@163.com		

实验目的：
实现 KNN，SVM，softmax 与三层神经网络

实验软件和硬件环境：
Vs code
Win10

实验原理和方法：

KNN：

就是找到一个样本与数据集中最相似的 k 个样本，然后选取这 k 个样本中最多的一个类别来表示这个样本的类别。

$$L_p(x_i, x_j) = \left(\sum_{l=1}^{n} |x_i^{(l)} - x_j^{(l)}|^p \right)^{\frac{1}{p}}$$

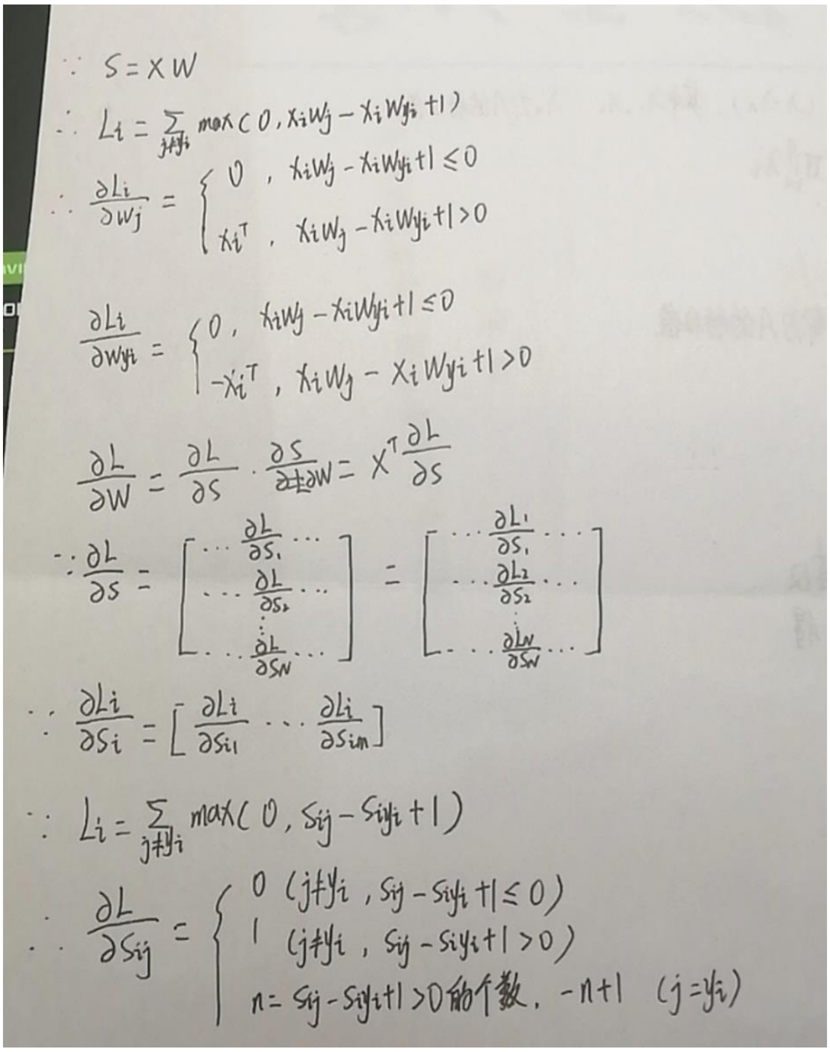
该实验中 p=2，计算的是欧式距离

SVM：

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_I} + 1)$$

$$L(W) = \frac{1}{N} \sum_{i=1}^{N} L_i(f(x_i; W), y_i) + \lambda R(W)$$

其中 R(W)使用的是 L2 正则化



$\because S = XW$

$\therefore L_i = \sum_{j \neq y_i} \max(0, x_i w_j - x_i w_{y_i} + 1)$

$\therefore \frac{\partial L_i}{\partial w_j} = \begin{cases} 0, & x_i w_j - x_i w_{y_i} + 1 \leq 0 \\ x_i^T, & x_i w_j - x_i w_{y_i} + 1 > 0 \end{cases}$

$\frac{\partial L_i}{\partial w_{y_i}} = \begin{cases} 0, & x_i w_j - x_i w_{y_i} + 1 \leq 0 \\ -x_i^T, & x_i w_j - x_i w_{y_i} + 1 > 0 \end{cases}$

$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial S} \cdot \frac{\partial S}{\partial W} = X^T \frac{\partial L}{\partial S}$

$\because \frac{\partial L}{\partial S} = \begin{bmatrix} \cdots & \frac{\partial L}{\partial S_1} & \cdots \\ \cdots & \frac{\partial L}{\partial S_2} & \cdots \\ & \vdots & \\ \cdots & \frac{\partial L}{\partial S_N} & \cdots \end{bmatrix} = \begin{bmatrix} \cdots & \frac{\partial L_1}{\partial S_1} & \cdots \\ \cdots & \frac{\partial L_2}{\partial S_2} & \cdots \\ & \vdots & \\ \cdots & \frac{\partial L_N}{\partial S_N} & \cdots \end{bmatrix}$

$\because \frac{\partial L_i}{\partial S_i} = \left[\frac{\partial L_i}{\partial S_{i1}} \cdots \frac{\partial L_i}{\partial S_{im}} \right]$

$\because L_i = \sum_{j \neq y_i} \max(0, S_{ij} - S_{iy_i} + 1)$

$\therefore \frac{\partial L}{\partial S_{ij}} = \begin{cases} 0 & (j \neq y_i,\ S_{ij} - S_{iy_i} + 1 \leq 0) \\ 1 & (j \neq y_i,\ S_{ij} - S_{iy_i} + 1 > 0) \\ n = S_{ij} - S_{iy_i} + 1 > 0 \text{的个数},\ -n+1 & (j = y_i) \end{cases}$

Softmax：

$$s = f(x_i; W) \qquad P(Y = k | X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$$

$$L_i = -logP(Y = y_i | X = x_i)$$

$\because S_{ij} = X_i \cdot W_j^T$

$softmax(S_{ij}) = \frac{e^{S_{ij}}}{\sum_k e^{S_{ik}}} = q_{ij}$

$L_i = -\log q_{im}$

$\therefore \frac{\partial L_i}{\partial W_j} = \frac{\partial L_i}{\partial q_i} \cdot \frac{\partial q_i}{\partial S_i} \cdot \frac{\partial S_i}{\partial W_j}$

$\because \frac{\partial L_i}{\partial q_i} = [0 \cdots, -\frac{1}{q_{i,y_i}} \cdots 0]$

$$\frac{\partial q_i}{\partial S_i} = \begin{bmatrix} \ddots & & \vdots \\ & \frac{\partial q_{im}}{\partial S_{in}} & \\ \cdots & & \ddots \end{bmatrix}$$

$$\therefore \frac{\partial q_{im}}{\partial S_{in}} = \begin{cases} m=n, & \frac{e^{S_{im}} \sum_k e^{S_{ik}} - e^{S_{im}} \cdot e^{S_{im}}}{(\sum_k e^{S_{ik}})^2} = q_{im}(1-q_{im}) \\ m \neq n, & \frac{-e^{S_{im}} \cdot e^{S_{in}}}{(\sum_k e^{S_{ik}})^2} = -q_{im} \cdot q_{in} \end{cases}$$

$$\frac{\partial S_i}{\partial W_j} = \begin{bmatrix} \cdots & 0 & \cdots \\ & x_i & \\ \cdots & 0 & \cdots \end{bmatrix} \text{第j行}$$

$$\therefore \frac{\partial L_i}{\partial W_j} = [0 \cdots, -\frac{1}{q_{i,y_i}} \cdots 0] \begin{bmatrix} \ddots & & \\ & \frac{\partial q_{im}}{\partial S_{in}} & \\ & & \ddots \end{bmatrix} \begin{bmatrix} \cdots 0 \\ \vdots \\ x_i \\ \vdots \\ \cdots 0 \end{bmatrix}$$

$$= [q_{i1}, q_{i2} \cdots q_{iy_i}-1 \cdots q_{in}] \begin{bmatrix} 0 \\ \vdots \\ x_i \\ 0 \end{bmatrix}$$

$$= \begin{cases} x_i \, q_{ij}, & j \neq y_i \\ x_i(q_{iy_i}-1), & j = y_i \end{cases}$$

$$\because \frac{\partial L}{\partial W} = \frac{\partial L}{\partial S} \cdot \frac{\partial S}{\partial W} = X^T \cdot \frac{\partial L}{\partial S}$$

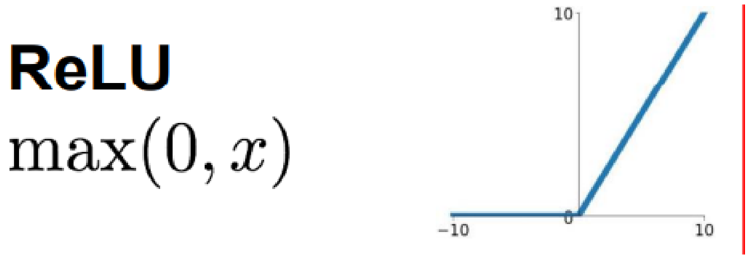
$$\because \frac{\partial L}{\partial S} = \begin{bmatrix} \cdots & \frac{\partial L_1}{\partial S_1} & \cdots \\ & \vdots & \\ \cdots & \frac{\partial L_i}{\partial S_i} & \cdots \end{bmatrix}$$

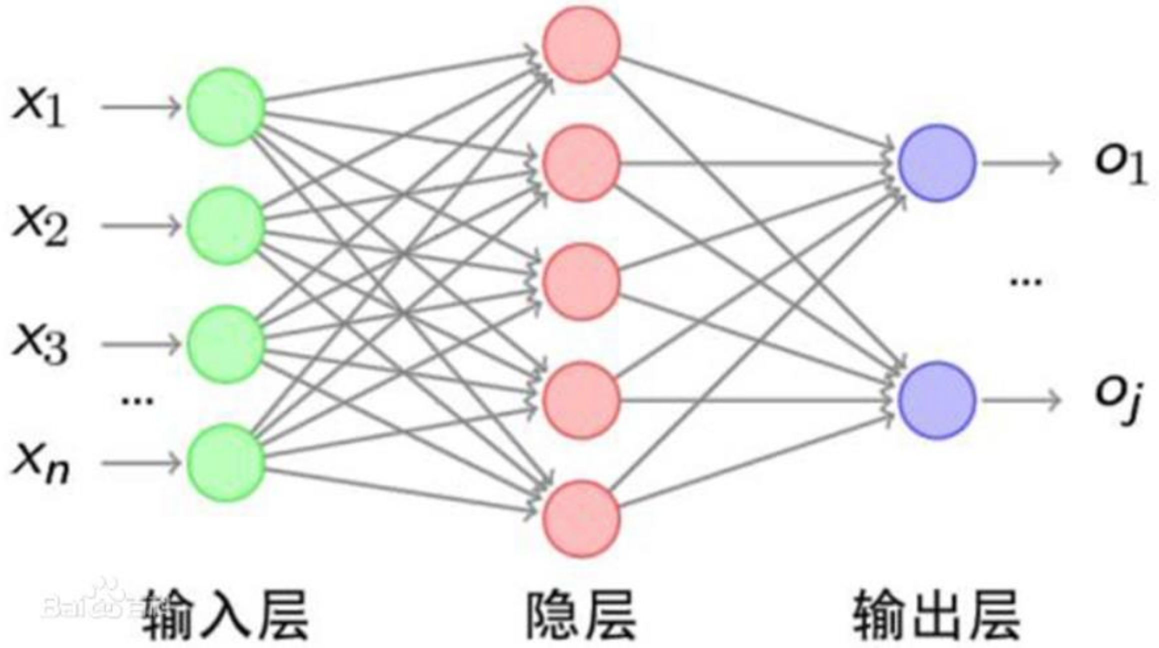
$$\because \frac{\partial L_i}{\partial S_i} = \frac{\partial L_i}{\partial q_i} \cdot \frac{\partial q_i}{\partial S_i}$$

$$= [0 \cdots \frac{-1}{q_{i,y_i}} \cdots 0] \begin{bmatrix} \ddots & & \\ & \frac{\partial q_{im}}{\partial S_{in}} & \\ & & \ddots \end{bmatrix}$$

$$= [q_{i1}, q_{i2} \cdots q_{iy_i}-1 \cdots q_{in}]$$

三层神经网络：





实验步骤：（不要求罗列完整源代码）

KNN：

```python
def compute_distances_two_loops(self, X):
    """
    Compute the distance between each test point in X and each training point
    in self.X_train using a nested loop over both the training data and the
    test data.

    Inputs:
    - X: A numpy array of shape (num_test, D) containing test data.

    Returns:
    - dists: A numpy array of shape (num_test, num_train) where dists[i, j]
      is the Euclidean distance between the ith test point and the jth training
      point.
    """
    num_test = X.shape[0]
    num_train = self.X_train.shape[0]
    dists = np.zeros((num_test, num_train))
    for i in range(num_test):
        for j in range(num_train):
            #####################################################################
            # TODO:                                                             #
            # Compute the l2 distance between the ith test point and the jth    #
            # training point, and store the result in dists[i, j]. You should   #
            # not use a loop over dimension, nor use np.linalg.norm().          #
            #####################################################################
            # *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****

            dists[i,j]=np.sqrt(np.sum(np.square(X[i,:]-self.X_train[j,:])))
            pass
            # *****END OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
    return dists
```

```python
def compute_distances_one_loop(self, X):
    """
    Compute the distance between each test point in X and each training point
    in self.X_train using a single loop over the test data.

    Input / Output: Same as compute_distances_two_loops
    """
    num_test = X.shape[0]
    num_train = self.X_train.shape[0]
    dists = np.zeros((num_test, num_train))
    for i in range(num_test):
        #######################################################################
        #                                                                     #
        # Compute the l2 distance between the ith test point and all training #
        # points, and store the result in dists[i, :].                        #
        # Do not use np.linalg.norm().                                        #
        #######################################################################
        # *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****

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

        #运用了广播的概念
        # *****END OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
    return dists
```

```
sum1=np.sum(np.square(X),axis=1)
sum2=np.sum(np.square(self.X_train),axis=1)
dot12=-2*np.dot(X,self.X_train.T)
dists=np.sqrt(sum1.reshape(-1,1)+sum2.T+dot12)
#print(dists.shape)
# *****END OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
return dists
```

计算欧式距离的方法有三种，分别是两重 for 循环，一重 for 循环和没有 for 循环，其中两重 for 循环是直接计算，而一重 for 循环则是使用了广播的概念，大体上和前者差不多，而没有循环的则是直接将最后的结果矩阵表示出来，三者相比较，运行时间 没有<两重<一重

```
y_1=np.argsort(dists[i,:]);
closest_y=self.y_train[y_1[:k]]

# *****END OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
#########################################################################
#                                                                       #
# Now that you have found the labels of the k nearest neighbors, you    #
# need to find the most common label in the list closest_y of labels.   #
# Store this label in y_pred[i]. Break ties by choosing the smaller     #
# label.                                                                #
#########################################################################
# *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****

y_pred[i]=argmax(np.bincount(closest_y))
# *****END OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
```

投票是先将一个同一个测试集的结果排序，然后选出最大的 k 个，然后选择其中最多的种类

```
X_train_folds=np.array_split(X_train,num_folds,axis=0)
y_train_folds=np.array_split(y_train,num_folds,axis=0)
```

```
for k in k_choices:
    accuracy_=[]
    for i in range(num_folds):
        xt=np.concatenate((X_train_folds[:i]+X_train_folds[i+1:]),axis=0)
        yt=np.concatenate((y_train_folds[:i]+y_train_folds[i+1:]),axis=0)
        classifier.train(xt,yt)
        yl=classifier.predict(X_train_folds[i],k=k)
        accuracy=np.mean(yl==y_train_folds[i])
        accuracy_+=[accuracy]
    k_to_accuracies[k]=accuracy_
pass
```

先将训练集分成 k 部分，然后在 for 循环中将第 i 个取出来作为测试数据，将其他的合为训练集，然后进行训练，之后将测试数据放进去，计算准确率

SVM：

```python
    # *****START OF YOUR CODE (DO NOT D
    dW /= num_train
    dW += 2 * reg * W
    pass
```

对于简单地计算 loss 和梯度，linear_svm 中已经求出来了，只需要将 dW 得出平均值，然后加上正则化地求导即可

```python
correct_scores = scores[np.arange(num_train), y]
correct_scores = correct_scores.reshape((num_train,-1))
margins = scores - correct_scores + 1
# print(margins)
margins[margins<=0]=0
loss += np.sum(margins)-num_train
loss /= num_train
loss += reg*np.sum(W*W)
pass

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

#############################################################################
# TODO:                                                                     #
# Implement a vectorized version of the gradient for the structured SVM     #
# loss, storing the result in dW.                                           #
#                                                                           #
# Hint: Instead of computing the gradient from scratch, it may be easier    #
# to reuse some of the intermediate values that you used to compute the     #
# loss.                                                                     #
#############################################################################
# *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
mask_ones = np.zeros_like(scores)
mask_ones[margins>0]=1
mask_ones[np.arange(num_train),y]=0
mask_ones[np.arange(num_train),y]=-np.sum(mask_ones,axis=1)+1
dW = 1/num_train * np.dot(X.T,mask_ones) + 2*reg*W
```

Loss 只需要求出 max 函数中全部不为 0 的位置的 scores 的和，但是因为在 j=yi 时，每个结果都是 1,所以要把前面的结果减去训练集的数据的个数,即减去 num_train,然后除以 num_train,再加上正则项即可

对于梯度，只需要将 max 函数大于 0 的位置置为 1，其它位置置为 0，然后 j=yi 的位置减去 max 函数大于 0 的个数即可，最后得到的矩阵再除以 num_train，再加上正则项的导数即可

实现 SGD 和交叉验证

```python
for it in range(num_iters):
    X_batch = None
    y_batch = None

    #########################################################################
    # TODO:                                                                 #
    # Sample batch_size elements from the training data and their           #
    # corresponding labels to use in this round of gradient descent.        #
    # Store the data in X_batch and their corresponding labels in           #
    # y_batch; after sampling X_batch should have shape (batch_size, dim)   #
    # and y_batch should have shape (batch_size,)                           #
    #                                                                       #
    # Hint: Use np.random.choice to generate indices. Sampling with         #
    # replacement is faster than sampling without replacement.              #
    #########################################################################
    # *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
    index = np.random.choice(num_train, batch_size, replace=True)
    X_batch = X[index]
    y_batch = y[index]

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

    # evaluate loss and gradient
    loss, grad = self.loss(X_batch, y_batch, reg)
    loss_history.append(loss)

    # perform parameter update
    #########################################################################
    # TODO:                                                                 #
    # Update the weights using the gradient and the learning rate.          #
    #########################################################################
    # *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
    self.W-=learning_rate*grad
    pass
```

```python
for rs in regularization_strengths:
    for lr in learning_rates:
        svm=LinearSVM()
        loss_hist=svm.train(X_train,y_train,lr,rs,num_iters=1500)
        y_train_pred=svm.predict(X_train)
        train_accuracy=np.mean(y_train==y_train_pred)
        y_val_pred=svm.predict(X_val)
        val_accuracy=np.mean(y_val==y_val_pred)
        if val_accuracy>best_val:
            best_val=val_accuracy
            best_svm=svm
        results[(lr,rs)]=train_accuracy,val_accuracy
```

Softmax

```python
# *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
for i in range(X.shape[0]):
    score=np.dot(X[i],W)
    score-=max(score)
    #print(score)
    score=np.exp(score)
    softmax_=np.sum(score)
    score/=softmax_
    loss-=np.log(score[y[i]])
    for j in range(W.shape[1]):
        if j==y[i]:
            dW[:,j]+=(score[j]-1)*X[i]
        else:
            dW[:,j]+=score[j]*X[i]


loss/=X.shape[0]
dW/=X.shape[0]
loss+=reg*np.sum(W*W)
dW+=2*reg*W

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

return loss, dW
```

为了防止数值溢出，减去每一行数据的最大值，然后只需要套公式就行

```python
# *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
scores=np.dot(X,W)
scores-=np.max(scores,axis=1,keepdims=True)
scores=np.exp(scores)
scores/=np.sum(scores,axis=1,keepdims=True)
# print(y)
loss=scores[np.arange(X.shape[0]),y]
# print(scores)
# print(scores[1][0])
# print(loss)
sum_loss=0
for i in range(loss.shape[0]):
    sum_loss-=np.log(loss[i])
loss=sum_loss/X.shape[0]
loss+=reg*np.sum(W*W)

ds=np.copy(scores)
# print(ds)
# print(scores)
ds[np.arange(X.shape[0]),y]-=1
dW=np.dot(X.T,ds)
dW/=X.shape[0]
dW+=2*reg*W
# *****END OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
```

和上面的差不多，先减去每行的最大值，然后套公式即可

三层神经网络：

```python
# *****START OF YOUR CODE (DO NOT DELETE/MOD
s1=np.dot(X,W1)+b1
s1_act=(s1>0)*s1
s2=np.dot(s1_act,W2)+b2
s2_act=(s2>0)*s2
scores=np.dot(s2_act,W3)+b3
pass
```

直接按照三层神经网络的结构来进行正向传播，然后得到结果矩阵

```python
# Compute the loss
loss = None
#############################################################################
# TODO: Finish the forward pass, and compute the loss. This should include  #
# both the data loss and L2 regularization for W1 and W2. Store the result  #
# in the variable loss, which should be a scalar. Use the Softmax           #
# classifier loss.                                                          #
#############################################################################
# *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
scores-=np.max(scores,axis=1,keepdims=True)
scores=np.exp(scores)
scores/=np.sum(scores,axis=1,keepdims=True)
loss=scores[np.arange(X.shape[0]),y]
sum_loss=0
for i in range(loss.shape[0]):
    sum_loss-=np.log(loss[i])
loss=sum_loss/X.shape[0]
loss+=reg*np.sum(W1*W1)+reg*np.sum(W2*W2)+reg*np.sum(W3*W3)
```

Loss 使用 softmax 来求，直接复制粘贴之前的代码即可

```python
# Backward pass: compute gradients
grads = {}
#############################################################################
# TODO: Compute the backward pass, computing the derivatives of the weights #
# and biases. Store the results in the grads dictionary. For example,       #
# grads['W1'] should store the gradient on W1, and be a matrix of same size #
#############################################################################
# *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
ds3=np.copy(scores)
ds3[np.arange(X.shape[0]),y]-=1
ds3/=X.shape[0]
#print(s1_act)
grads['W3']=np.dot(s2_act.T,ds3)+2*reg*W3
grads['b3']=np.sum(ds3,axis=0)

ds2=np.dot(ds3,W3.T)
ds2=(s2>0)*ds2
grads['W2']=np.dot(s1_act.T,ds2)+2*reg*W2
grads['b2']=np.sum(ds2,axis=0)

ds1=np.dot(ds2,W2.T)
ds1=(s1>0)*ds1
grads['W1']=np.dot(X.T,ds1)+2*reg*W1
grads['b1']=np.sum(ds1,axis=0)

#ds2=np.dot()
pass
```

W1，b1 之类的数据则使用反向传播来求出公式，然后计算即可

```python
best_acc = -1
#################################################################################
# TODO: Tune hyperparameters using the validation set. Store your best trained  #
# model in best_net.                                                            #
#                                                                               #
# To help debug your network, it may help to use visualizations similar to the  #
# ones we used above; these visualizations will have significant qualitative    #
# differences from the ones we saw above for the poorly tuned network.          #
#                                                                               #
# Tweaking hyperparameters by hand can be fun, but you might find it useful to  #
# write code to sweep through possible combinations of hyperparameters          #
# automatically like we did on the previous exercises.                          #
#################################################################################
learning_rate = [1e-4, 5e-3,1e-3,1e-2]
regularization = [1e-4,5e-4,1e-3,4e-3,1e-2]

for lr in learning_rate:
    for reg in regularization:
        net = ThreeLayerNet(input_size, hidden_size, num_classes)
        state = net.train(X_train, y_train, X_val, y_val,
                    num_iters=10000, batch_size=150,
                    learning_rate=lr, learning_rate_decay=0.95,
                    reg=reg, verbose=False)
        val_acc = np.mean(net.predict(X_val) == y_val)
        if val_acc > best_acc:
            best_acc = val_acc
            best_net = deepcopy(net)
print('best val acc: {:.3f}'.format(best_acc))
#################################################################################
#                              END OF YOUR CODE                                 #
#################################################################################
```

结论分析与体会：

这个实验很大，将之前上课所说的知识都过了一遍，按照代码补全差不多都能得到它要求的的结果，而且向量化的速度会比使用 for 循环快很多，其中最难的还是梯度推导的部分。

就实验过程中遇到和出现的问题，你是如何解决和处理的，自拟 1—3 道问答题：

1. 各种梯度的计算其实都不是很懂，因为要考虑到矩阵的维度，所以会很麻烦。
2. 在三层神经网络部分，在计算 loss 的时候，使用的是 softmax，但是最后得到的结果和它要求的差了 0.04 左右，精度差了很多，后面换了 svm 差了 0.9 左右，精度差了更多，所以这个问题依旧没有解决。
3. 在使用三层神经网络模型进行测试的时候，因为参数设置的不好，迭代次数太低，所以得到的结果不好，后面看群里的消息，将迭代次数改为了 10000，就能得到了

很好的准确率，达到了 50%以上。