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：Homework_1		学号：201900301174
日期：2021.10.2	班级： 智能19	姓名： 韩旭
Email：hanx@mail.sdu.edu.cn		
实验目的： In this assignment you will practice putting together a simple image classification pipeline, based on the k-Nearest Neighbor or the SVM/Softmax classifier.		
实验软件和硬件环境： Ubuntu20.04 RTX3090 Jupyter notebook		
实验原理和方法： The goals of this assignment are as follows: • understand the basic Image Classification pipeline and the data-driven approach (train/predict stages) • understand the train/val/test splits and the use of validation data for hyperparameter tuning. • develop proficiency in writing efficient vectorized code with numpy • implement and apply a k-Nearest Neighbor (kNN) classifier • implement and apply a Multiclass Support Vector Machine (SVM) classifier • implement and apply a Softmax classifier • implement and apply a Three layer neural network classifier • understand the differences and tradeoffs between these classifiers • get a basic understanding of performance improvements from using higher-level representations than raw pixels (e.g. color histograms, Histogram of Gradient (HOG) features)		

实验步骤：（不要求罗列完整源代码）

KNN：

k_nearest_neighbor.py:

两次循环：

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

一次循环：

```
# pass
dists[i][:] = np.sqrt(np.sum(np.square(self.X_train - X[i, :]), axis=1))
# *****END OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
```

没有循环：使用 reshape、点乘

```
        # pass
        dists = np.sqrt(np.reshape(np.sum(np.square(X), axis=1), (X.shape[0], 1)) +
 np.sum(np.square(self.X_train),
                                                                    axis=1)
 - 2 * X.dot(self.X_train.T))
        # *****END OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
```

预测函数：

```
# pass
closest_y = list(self.y_train[np.argsort(dists[i])[0:k]])
# *****END OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
```

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

Knn.ipynb

2 Inline Question 1

Notice the structured patterns in the distance matrix, where some rows or columns are visible brighter. (Note that with the default color scheme black indicates low distances while white indicates high distances.)

- What in the data is the cause behind the distinctly bright rows?
- What causes the columns?

Your Answer：1,每一行表示每一个测试数据与所有训练数据的距离，明显偏亮的那几行测试数据与所有的训练数据差别都较大，即训练数据中没有出现过这几类样本或者这几类与其他类的差别较大，导致测试数据与训练数据距离偏大，于是那几行偏亮。2,每一列表示每一个训练数据与所有测试数据的距离，偏亮的那几个训练数据与所有测试数据的距离都偏大，于是那几列偏亮。

Inline Question 2

We can also use other distance metrics such as L1 distance. For pixel values $p_{ij}^{(k)}$ at location (i, j) of some image I_k,

the mean μ across all pixels over all images is

$$\mu = \frac{1}{nhw} \sum_{k=1}^{n} \sum_{i=1}^{h} \sum_{j=1}^{w} p_{ij}^{(k)}$$

And the pixel-wise mean μ_{ij} across all images is

$$\mu_{ij} = \frac{1}{n} \sum_{k=1}^{n} p_{ij}^{(k)}.$$

The general standard deviation σ and pixel-wise standard deviation σ_{ij} is defined similarly.

Which of the following preprocessing steps will not change the performance of a Nearest Neighbor classifier that uses L1 distance? Select all that apply.

1. Subtracting the mean μ ($\tilde{p}_{ij}^{(k)} = p_{ij}^{(k)} - \mu$.)
2. Subtracting the per pixel mean μ_{ij} ($\tilde{p}_{ij}^{(k)} = p_{ij}^{(k)} - \mu_{ij}$.)
3. Subtracting the mean μ and dividing by the standard deviation σ.
4. Subtracting the pixel-wise mean μ_{ij} and dividing by the pixel-wise standard deviation σ_{ij}.
5. Rotating the coordinate axes of the data.

Your Answer：*1，2，3*

Your Explanation：*1，|| (x - m) - (y - m) || = || x - m - y + m || = || x - y ||没有变 2，|| (x - s) - (y - s) || = || x - s - y + s || = || x - y ||没有变 3，|| (x/s) - (y/s) || = || (1/s) (x - y) || = (1/s) || x - y ||乘了个相同的系数，等价于没变*

交叉验证：

分数据：

```
#pass
X_train_folds = np.array_split(X_train, num_folds)
y_train_folds = np.array_split(y_train, num_folds)


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

K 折交叉验证过程

```
#pass
for k in k_choices:
    k_to_accuracies[k] = []
    for i in range(num_folds):
        X_test_flods=X_train_folds[i]
        y_test_flods=y_train_folds[i]
        X_train_f=np.concatenate(X_train_folds[0:i]+X_train_folds[i+1:])
        y_train_f=np.concatenate(y_train_folds[0:i]+y_train_folds[i+1:])
        classifier.train(X_train_f,y_train_f)
        dis=classifier.compute_distances_no_loops(X_test_flods)
        y_test_pred=classifier.predict_labels(dis,k)
        num_yes=np.sum(y_test_pred==y_test_flods)
        k_to_accuracies[k].append(float(num_yes/y_test_flods.shape[0]))
# *****END OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
```

中间结果截图

```
k = 20, accuracy = 0.270000
k = 20, accuracy = 0.279000
k = 20, accuracy = 0.279000
k = 20, accuracy = 0.282000
k = 20, accuracy = 0.285000
k = 50, accuracy = 0.271000
k = 50, accuracy = 0.288000
k = 50, accuracy = 0.278000
k = 50, accuracy = 0.269000
k = 50, accuracy = 0.266000
k = 100, accuracy = 0.256000
k = 100, accuracy = 0.270000
k = 100, accuracy = 0.263000
k = 100, accuracy = 0.256000
k = 100, accuracy = 0.263000
```

结果：

```
# Compute and display the accuracy
num_correct = np.sum(y_test_pred == y_test)
accuracy = float(num_correct) / num_test
print('Got %d / %d correct => accuracy: %f' % (num_correct, num_test, accuracy))
```

executed in 276ms, finished 16:50:28 2021-10-04

```
Got 141 / 500 correct => accuracy: 0.282000
```

Inline Question 3

Which of the following statements about k-Nearest Neighbor (k-NN) are true in a classification setting, and for all k? Select all that apply.

1. The decision boundary of the k-NN classifier is linear.
2. The training error of a 1-NN will always be lower than that of 5-NN.
3. The test error of a 1-NN will always be lower than that of a 5-NN.
4. The time needed to classify a test example with the k-NN classifier grows with the size of the training set.
5. None of the above.

Your Answer: *4*

Your Explanation: *因为在训练时要处理所有训练数据，所以训练数据变多了时间自然变长*

SVM：

linear_svm.py：

svm_loss_naïve 方法

```
# pass
dW/=num_train
dW+=2*reg*W

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

svm_loss_vectorized 方法

```
#pass
scores=X.dot(W)
N=X.shape[0]
correct_class_scores=scores[range(N),y]
margin=scores-scores[range(0,N),y].reshape(-1,1)+1
margin[range(N),y]=0
margin=(margin>0)*margin
loss+=margin.sum()/N
loss+=reg * np.sum(W * W)

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

Hinge loss

```
#pass
counts = (margin > 0).astype(int)
counts[range(N), y] = - np.sum(counts, axis = 1)
dW += np.dot(X.T, counts) / N
dW+=2*reg * W

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

测试结果：

```
In [24]:
    # Evaluate the naive implementation of the loss we provided for you:
    from sducs2019.classifiers.linear_svm import svm_loss_naive
    import time

    # generate a random SVM weight matrix of small numbers
    W = np.random.randn(3073, 10) * 0.0001

    loss, grad = svm_loss_naive(W, X_dev, y_dev, 0.000005)
    print('loss: %f' % (loss, ))
```

executed in 192ms, finished 19:27:34 2021-10-04

```
loss: 9.467782
```

Inline Question1：

Inline Question 1

It is possible that once in a while a dimension in the gradcheck will not match exactly. What could such a discrepancy be caused by? Is it a reason for concern? What is a simple example in one dimension where a gradient check could fail? How would change the margin affect of the frequency of this happening? *Hint: the SVM loss function is not strictly speaking differentiable*

Your Answer：*这个数值解是近似解，当损失函数在某些点不可导的时候，例如零点，两个值就会出现差距。*

Next implement the function svm_loss_vectorized; for now only compute the loss;
we will implement the gradient in a moment.

```
Naive loss: 9.467782e+00 computed in 0.190736s
Vectorized loss: 9.467782e+00 computed in 0.004533s
difference: -0.000000
```

Complete the implementation of svm_loss_vectorized, and compute the gradient of the loss function in a vectorized way.

```
Naive loss and gradient: computed in 0.184144s
Vectorized loss and gradient: computed in 0.003100s
difference: 0.000000
```

linear_classifier.py

train 方法

```
#pass
ind=np.random.choice(num_train,batch_size)
X_batch=X[ind]
y_batch=y[ind]

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

```
#pass
self.W-=learning_rate*grad

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

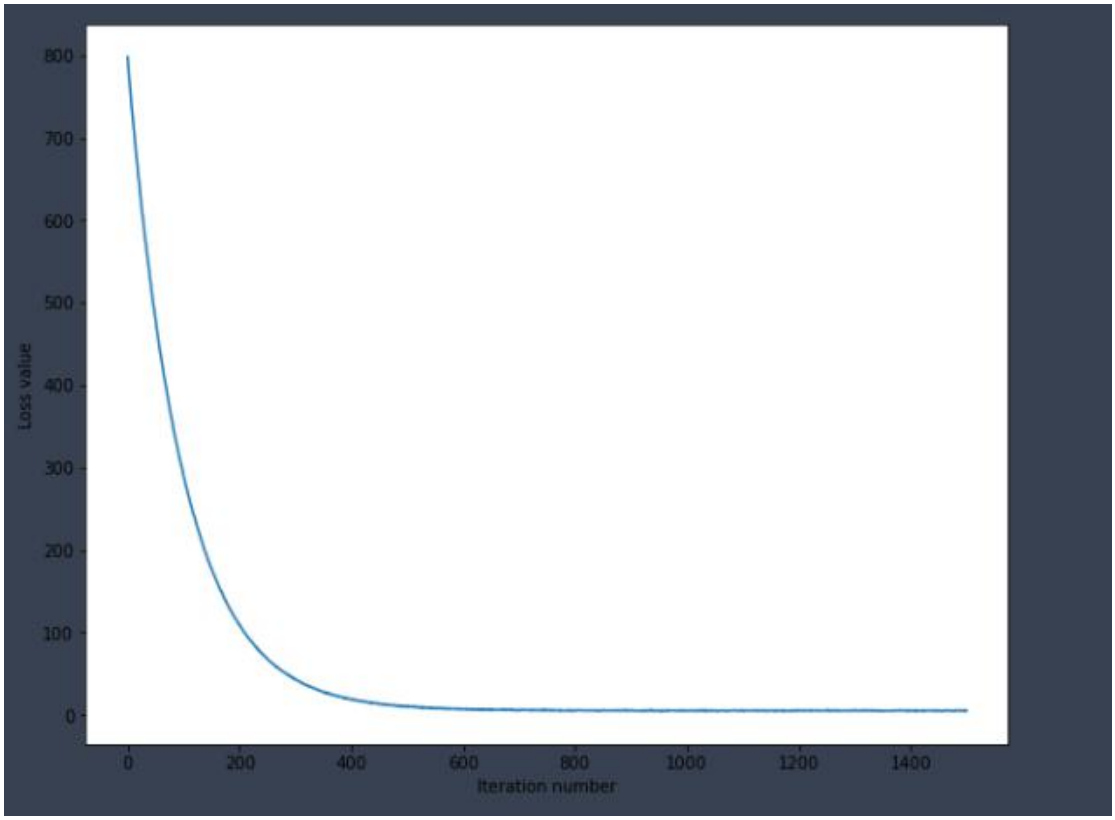
Predict 方法

```
#pass
scores=np.dot(X,self.W)
y_pred=np.argmax(scores,axis=1)

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

In the file linear_classifier.py, implement SGD in the function LinearClassifier.train() and then run it with the code below.

```
iteration 0 / 1500: loss 798.143299
iteration 100 / 1500: loss 289.118323
iteration 200 / 1500: loss 108.677067
iteration 300 / 1500: loss 43.241487
iteration 400 / 1500: loss 19.122957
iteration 500 / 1500: loss 10.811234
iteration 600 / 1500: loss 6.941845
iteration 700 / 1500: loss 5.492073
iteration 800 / 1500: loss 5.135653
iteration 900 / 1500: loss 5.656129
iteration 1000 / 1500: loss 5.249294
iteration 1100 / 1500: loss 5.573492
iteration 1200 / 1500: loss 5.680929
iteration 1300 / 1500: loss 5.691317
iteration 1400 / 1500: loss 5.135271
That took 5.103389s
```

plot the loss as a function of iteration number:



预测结果：

In [36]:

```
# Write the LinearSVM.predict function and evaluate the performance on
# training and validation set
y_train_pred = svm.predict(X_train)
print('training accuracy: %f' % (np.mean(y_train == y_train_pred), ))
y_val_pred = svm.predict(X_val)
print('validation accuracy: %f' % (np.mean(y_val == y_val_pred), ))
```

executed in 168ms, finished 20:04:17 2021-10-04

```
training accuracy: 0.366939
validation accuracy: 0.373000
```

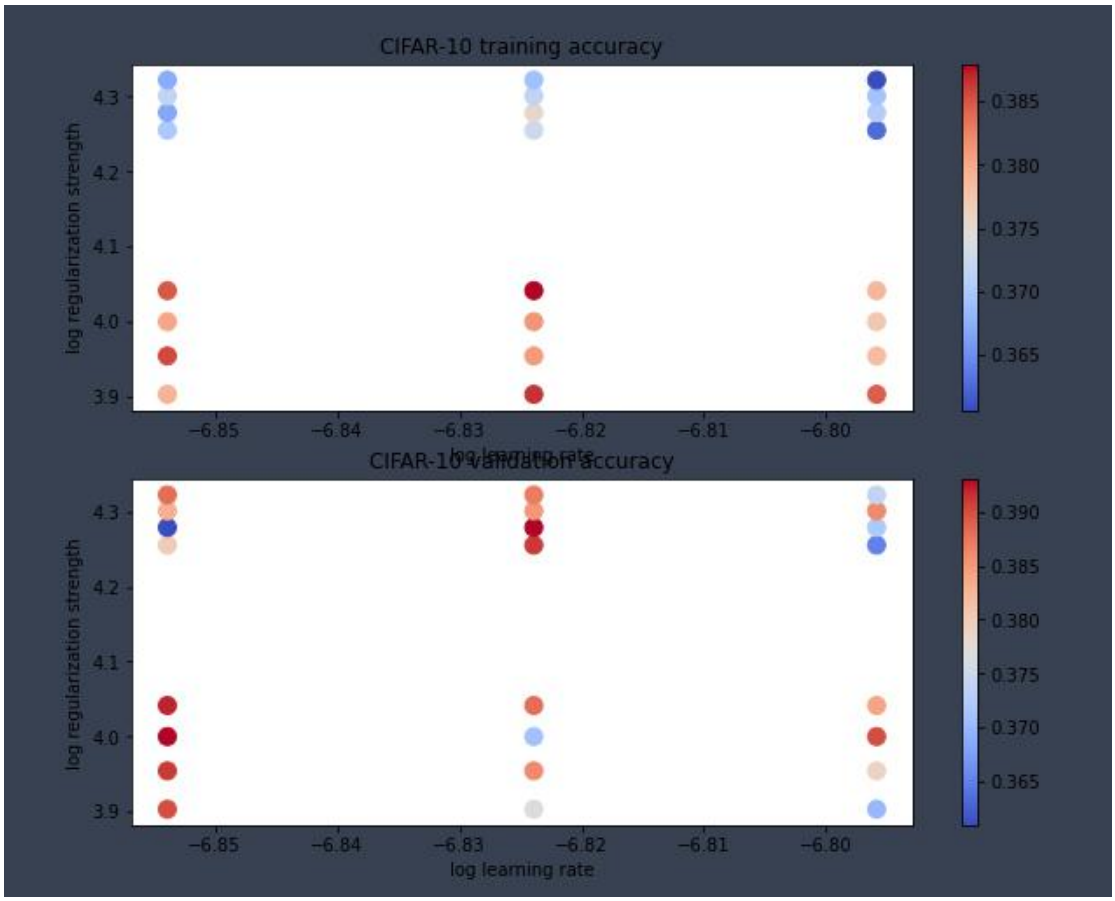
搜索超参：

```
#pass
for lr in learning_rates:
    for reg in regularization_strengths:
        classifier_svm=LinearSVM()
        loss_val=classifier_svm.train(X_train,y_train,learning_rate=lr
        train_acc=np.mean(classifier_svm.predict(X_train)==y_train)
        val_acc=np.mean(classifier_svm.predict(X_val)==y_val)
        results[(lr,reg)]=(train_acc,val_acc)
        if(val_acc>best_val):
            best_svm=classifier_svm
            best_val=val_acc

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

```
lr 1.400000e-07 reg 1.900000e+04 train accuracy: 0.366857 val accuracy: 0.361000
lr 1.400000e-07 reg 2.000000e+04 train accuracy: 0.371776 val accuracy: 0.383000
lr 1.400000e-07 reg 2.100000e+04 train accuracy: 0.367245 val accuracy: 0.388000
lr 1.500000e-07 reg 8.000000e+03 train accuracy: 0.386612 val accuracy: 0.377000
lr 1.500000e-07 reg 9.000000e+03 train accuracy: 0.380980 val accuracy: 0.386000
lr 1.500000e-07 reg 1.000000e+04 train accuracy: 0.381347 val accuracy: 0.371000
lr 1.500000e-07 reg 1.100000e+04 train accuracy: 0.387939 val accuracy: 0.388000
lr 1.500000e-07 reg 1.800000e+04 train accuracy: 0.372755 val accuracy: 0.391000
lr 1.500000e-07 reg 1.900000e+04 train accuracy: 0.375469 val accuracy: 0.393000
lr 1.500000e-07 reg 2.000000e+04 train accuracy: 0.371673 val accuracy: 0.385000
lr 1.500000e-07 reg 2.100000e+04 train accuracy: 0.369082 val accuracy: 0.387000
lr 1.600000e-07 reg 8.000000e+03 train accuracy: 0.384551 val accuracy: 0.370000
lr 1.600000e-07 reg 9.000000e+03 train accuracy: 0.378490 val accuracy: 0.379000
lr 1.600000e-07 reg 1.000000e+04 train accuracy: 0.377347 val accuracy: 0.390000
lr 1.600000e-07 reg 1.100000e+04 train accuracy: 0.378898 val accuracy: 0.384000
lr 1.600000e-07 reg 1.800000e+04 train accuracy: 0.362571 val accuracy: 0.365000
lr 1.600000e-07 reg 1.900000e+04 train accuracy: 0.370592 val accuracy: 0.372000
lr 1.600000e-07 reg 2.000000e+04 train accuracy: 0.369286 val accuracy: 0.386000
lr 1.600000e-07 reg 2.100000e+04 train accuracy: 0.360612 val accuracy: 0.374000
best validation accuracy achieved during cross-validation: 0.393000
```

plot training accuracy

plot validation accuracy



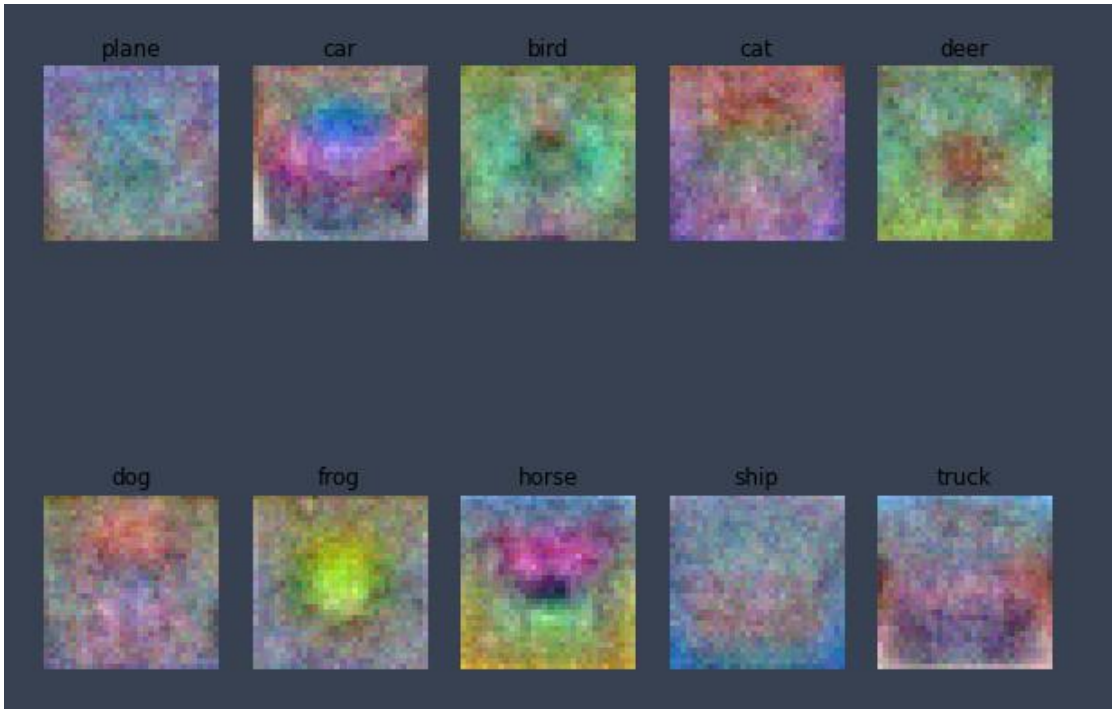
模型评价

```
# Evaluate the best svm on test set
y_test_pred = best_svm.predict(X_test)
test_accuracy = np.mean(y_test == y_test_pred)
print('linear SVM on raw pixels final test set accuracy: %f' % test_ac
```

executed in 12ms, finished 20:20:20 2021-10-04

```
linear SVM on raw pixels final test set accuracy: 0.380000
```

画出分类图：



Inline question 2

Describe what your visualized SVM weights look like, and offer a brief explanation for why they look they way that they do.

Your Answer：SVM分类的weight图就像是每一个类的比较总体比较模糊的特征分类器，每一张图片都是有噪音的，不过也是能看出大致的轮廓。比如汽车和卡车会有一个红色的汽车轮廓，青蛙会有一个模糊的绿色物体，鹿和狗会有一个棕黄色物体，船的背景是蓝色的水。

Softmax

```
# First implement the naive softmax loss function with nested loops.
# Open the file sducs2019/classifiers/softmax.py and implement the
# softmax_loss_naive function.

from sducs2019.classifiers.softmax import softmax_loss_naive
import time

# Generate a random softmax weight matrix and use it to compute the lo
W = np.random.randn(3073, 10) * 0.0001
loss, grad = softmax_loss_naive(W, X_dev, y_dev, 0.0)

# As a rough sanity check, our loss should be something close to -log(
print('loss: %f' % loss)
print('sanity check: %f' % (-np.log(0.1)))
```

executed in 141ms, finished 20:54:02 2021-10-04

```
loss: 2.366328
sanity check: 2.302585
```

softmax_loss_naïve 方法

```
#pass
num_train=X.shape[0]
for i in range(num_train):
    f =X[i].dot(W)
    f -= np.max(f) #logC=-maxjfj
    loss =loss+ np.log(np.sum(np.exp(f))) - f[y[i]]
    dW[:,y[i]]-=X[i]
    for j in range(W.shape[1]):
        dW[:,j]+=np.exp(f[j])/np.exp(f).sum()*X[i]
loss/=num_train
loss+=reg * np.sum(W*W)
dW /= num_train
dW += 2 * reg * W

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

Inline Question 1

Why do we expect our loss to be close to -log(0.1)? Explain briefly.**

Your Answer: *一共有十个类，最初scores都差不多，所以我们希望接近-log(0.1)*

Check 梯度：

In [5]:

```
# Complete the implementation of softmax_loss_naive and implement a (n
# version of the gradient that uses nested loops.
loss, grad = softmax_loss_naive(W, X_dev, y_dev, 0.0)

# As we did for the SVM, use numeric gradient checking as a debugging
# The numeric gradient should be close to the analytic gradient.
from sducs2019.gradient_check import grad_check_sparse
f = lambda w: softmax_loss_naive(w, X_dev, y_dev, 0.0)[0]
grad_numerical = grad_check_sparse(f, W, grad, 10)

# similar to SVM case, do another gradient check with regularization
loss, grad = softmax_loss_naive(W, X_dev, y_dev, 5e1)
f = lambda w: softmax_loss_naive(w, X_dev, y_dev, 5e1)[0]
grad_numerical = grad_check_sparse(f, W, grad, 10)
```

executed in 5.20s, finished 20:58:09 2021-10-04

```
numerical: -0.097817 analytic: -0.097817, relative error: 1.745586e-07
numerical: -0.445957 analytic: -0.445957, relative error: 6.773364e-09
numerical: -0.322940 analytic: -0.322940, relative error: 2.338420e-07
numerical: 3.255966 analytic: 3.255966, relative error: 2.418784e-08
numerical: -0.183707 analytic: -0.183707, relative error: 2.794433e-08
numerical: -0.003874 analytic: -0.003874, relative error: 7.369761e-07
numerical: -1.975530 analytic: -1.975530, relative error: 1.333111e-09
numerical: -0.001002 analytic: -0.001002, relative error: 1.686089e-05
numerical: 1.388580 analytic: 1.388580, relative error: 1.835913e-08
numerical: 3.811055 analytic: 3.811055, relative error: 1.279075e-08
numerical: -2.240419 analytic: -2.240419, relative error: 4.346691e-09
numerical: -0.132299 analytic: -0.132299, relative error: 2.152984e-07
numerical: -0.959965 analytic: -0.959965, relative error: 2.761503e-08
numerical: 3.441066 analytic: 3.441066, relative error: 8.766413e-10
numerical: -2.538300 analytic: -2.538300, relative error: 8.259178e-09
numerical: 0.377471 analytic: 0.377471, relative error: 2.051719e-07
numerical: 0.851385 analytic: 0.851384, relative error: 3.317408e-08
numerical: 1.416610 analytic: 1.416609, relative error: 2.182315e-08
numerical: -0.791157 analytic: -0.791158, relative error: 5.381347e-08
numerical: 0.883480 analytic: 0.883480, relative error: 1.085930e-08
```

softmax_loss_vectorized 方法：

```
#pass
num_train=X.shape[0]
f = np.dot(X, W)
f -= f.max(axis = 1).reshape(num_train, 1)
loss=np.log(np.exp(f).sum(axis=1)).sum()-f[range(num_train),y].sum()
p= np.exp(f) /np.exp(f).sum(axis = 1).reshape(num_train,1)
p[range(num_train),y]-=1
loss/=num_train
loss+=reg * np.sum(W * W)
dW=np.dot(X.T,p)/num_train
dW+=2*reg*W

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

结果对比：

```
naive loss: 2.366328e+00 computed in 0.131508s
vectorized loss: 2.366328e+00 computed in 0.003291s
Loss difference: 0.000000
Gradient difference: 0.000000
```

交叉搜索：

```
#pass
for lr in learning_rates:
    for reg in regularization_strengths:
        classifier_softmax=Softmax()
        loss_val=classifier_softmax.train(X_train,y_train,learning_rat
        train_acc=np.mean(classifier_softmax.predict(X_train)==y_train
        val_acc=np.mean(classifier_softmax.predict(X_val)==y_val)
        results[(lr,reg)]=(train_acc,val_acc)
        if(val_acc>best_val):
            best_softmax=classifier_softmax
            best_val=val_acc

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

中间结果

```
iteration 1200 / 1500: loss 2.150622
iteration 1300 / 1500: loss 2.171639
iteration 1400 / 1500: loss 2.176455
lr 1.000000e-07 reg 2.500000e+04 train accuracy: 0.331020 val accuracy: 0.351000
lr 1.000000e-07 reg 5.000000e+04 train accuracy: 0.314531 val accuracy: 0.327000
lr 5.000000e-07 reg 2.500000e+04 train accuracy: 0.323592 val accuracy: 0.327000
lr 5.000000e-07 reg 5.000000e+04 train accuracy: 0.308469 val accuracy: 0.319000
best validation accuracy achieved during cross-validation: 0.351000
```

模型评价

```
# evaluate on test set
# Evaluate the best softmax on test set
y_test_pred = best_softmax.predict(X_test)
test_accuracy = np.mean(y_test == y_test_pred)
print('softmax on raw pixels final test set accuracy: %f' % (test_accu
```

executed in 19ms, finished 21:00:44 2021-10-04

```
softmax on raw pixels final test set accuracy: 0.343000
```

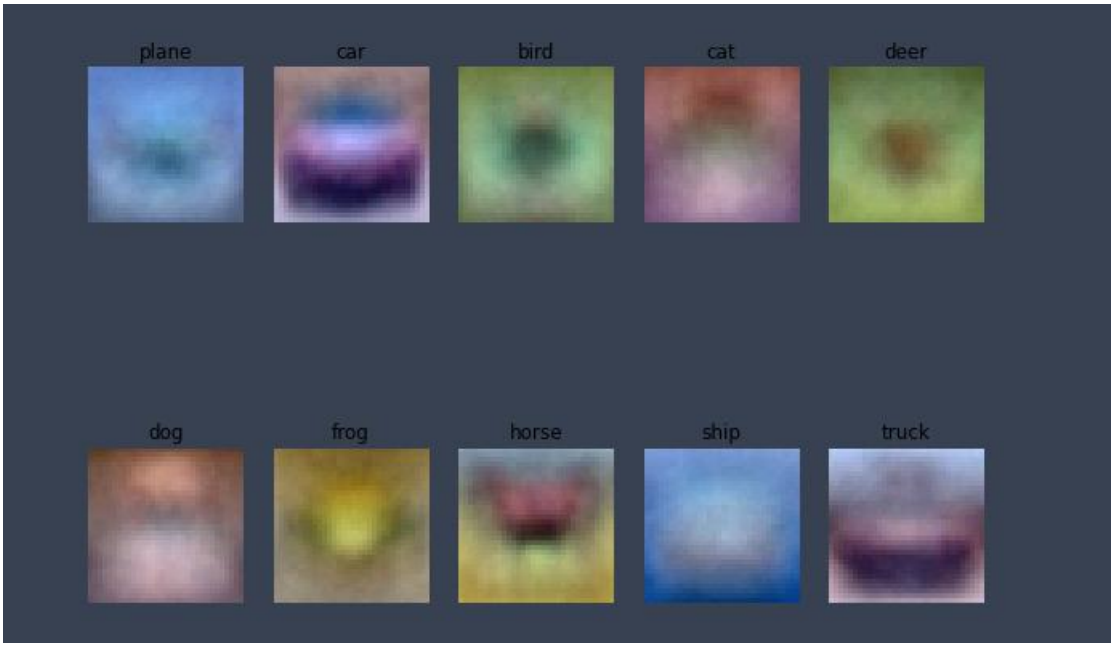
Inline Question 2 - *True or False*

Suppose the overall training loss is defined as the sum of the per-datapoint loss over all training examples. It is possible to add a new datapoint to a training set that would leave the SVM loss unchanged, but this is not the case with the Softmax classifier loss.

Your Answer : *True*

Your Explanation : *因为SVM求loss的时候会有max操作，如果新加的点求的是0，那么对loss没有改变。而softmax添加新的训练样本则会改变数据分布，loss肯定会改变*

权重图



三层网络：

Neural_net.py

Loss 方法，三层网络计算 loss，最后正则化

```
#pass
layer1=np.maximum(0, X.dot(W1) + b1)
layer2=np.maximum(0, layer1.dot(W2) + b2)
scores=layer2.dot(W3) + b3

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

```
        #pass
        scores -= np.max(scores, axis=1).reshape(-1,1)
        probabilitites = np.exp(scores) / np.sum(np.exp(scores), axis =
1).reshape(-1,1)
        loss = np.mean(-np.log(probabilitites[range(N), y]))
        loss += reg * np.sum(W1*W1)
        loss += reg * np.sum(W2*W2)
        loss += reg * np.sum(W3*W3)


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

误差反向传播过程

```
#pass
grad_Z3 = probabilitites
grad_Z3[range(N), y] -= 1.0
grad_Z3 /= N

grads["b3"] = np.sum(grad_Z3, axis = 0)
grads["W3"] = layer2.T.dot(grad_Z3)
grads["W3"] += 2*reg * W3


grad_O2 = grad_Z3.dot(W3.T)
grad_O2[layer2 <= 0] = 0 #the RELU function.

grads["b2"] = np.sum(grad_O2, axis = 0)
grads["W2"] = layer1.T.dot(grad_O2)
grads["W2"] += 2*reg * W2


grad_O1 = grad_O2.dot(W2.T)
grad_O1[layer1 <= 0] = 0 #the RELU function.

grads["b1"] = np.sum(grad_O1, axis = 0)
grads["W1"] = X.T.dot(grad_O1)
grads["W1"] += 2*reg * W1

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

Forward pass: compute scores

```
Your scores:
[[-0.03596154 -0.01613583 -0.00048556]
 [-0.09480192  0.20724618 -0.09763798]
 [-0.07015667  0.17081869 -0.07675745]
 [ 0.01473052  0.09321097 -0.0395178 ]
 [-0.05306489  0.04729807  0.01750587]]

correct scores:
[[-0.03596154 -0.01613583 -0.00048556]
 [-0.09480192  0.20724618 -0.09763798]
 [-0.07015667  0.17081869 -0.07675745]
 [ 0.01473052  0.09321097 -0.0395178 ]
 [-0.05306489  0.04729807  0.01750587]]

Difference between your scores and correct scores:
4.807962379632267e-08
```

Forward pass: compute loss

```
loss, _ = net.loss(X, y, reg=0.025)
correct_loss =  1.131301031558

# should be very small, we get < 1e-12
print('Difference between your loss and correct loss:')
print(np.sum(np.abs(loss - correct_loss)))
```

executed in 8ms, finished 14:53:35 2021-10-05

```
Difference between your loss and correct loss:
7.829292769656604e-13
```

Backward pass

```
In [5]:
from sducs2019.gradient_check import eval_numerical_gradient

# Use numeric gradient checking to check your implementation of the ba
# If your implementation is correct, the difference between the numeri
# analytic gradients should be less t 1e-8 for each of W1, W2, b1,

loss, grads = net.loss(X, y, reg=0.025)

# these should all be less than 1e-8 or so
for param_name in grads:
    f = lambda W: net.loss(X, y, reg=0.025)[0]
    param_grad_num = eval_numerical_gradient(f, net.params[param_name]
    print('%s max relative error: %e' % (param_name, rel_error(param_g
```

executed in 54ms, finished 14:53:35 2021-10-05

```
b3 max relative error: 8.494237e-11
W3 max relative error: 1.067978e-08
b2 max relative error: 4.590211e-10
W2 max relative error: 2.542857e-08
b1 max relative error: 7.670389e-09
W1 max relative error: 3.561318e-09
```

Train the network

Train 方法

```
#pass
indices = np.random.choice(num_train, batch_size, replace=True)
X_batch = X[indices]
y_batch = y[indices]

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

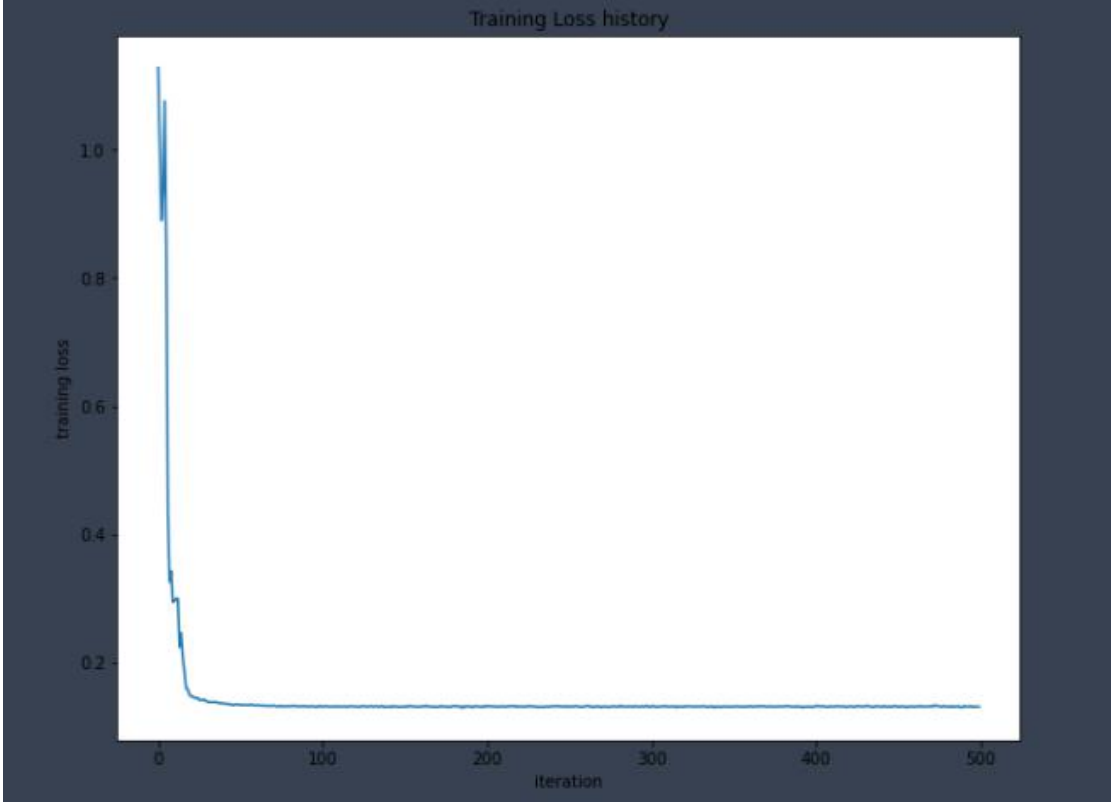
```
#pass
self.params['W1'] -= learning_rate * grads['W1']
self.params['b1'] -= learning_rate * grads['b1']
self.params['W2'] -= learning_rate * grads['W2']
self.params['b2'] -= learning_rate * grads['b2']
self.params['W3'] -= learning_rate * grads['W3']
self.params['b3'] -= learning_rate * grads['b3']

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

Predict 方法

```python
#pass
W1, b1 = self.params['W1'], self.params['b1']
W2, b2 = self.params['W2'], self.params['b2']
W3, b3 = self.params['W3'], self.params['b3']
layer1 = np.maximum(0, np.dot(X, W1) + b1)
layer2 = np.maximum(0, np.dot(layer1, W2) + b2)
scores = np.dot(layer2, W3) + b3
y_pred = np.argmax(scores, axis = 1)

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

结果图：

```
Final training loss:  0.13032214114367924
```



7 Train a network

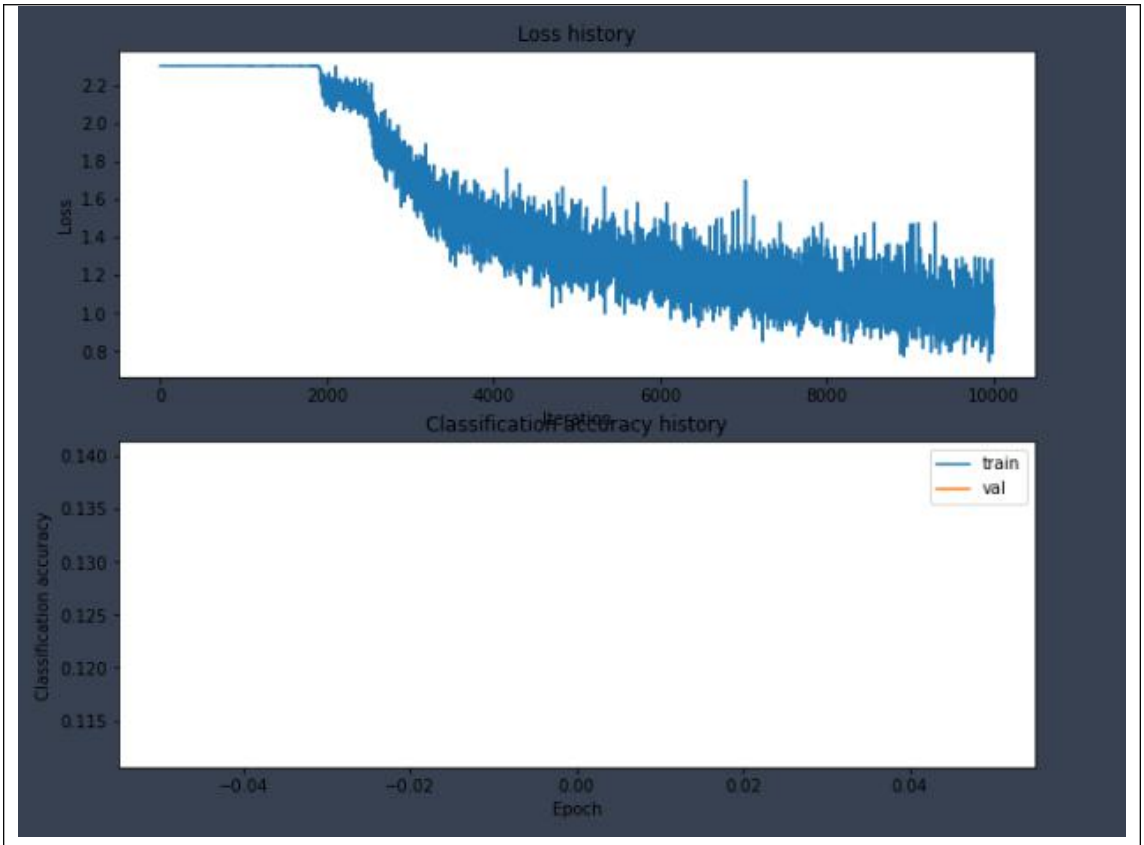
To train our network we will use SGD. In addition, we will adjust the learning rate with an exponential learning rate schedule as optimization proceeds; after each epoch, we will reduce the learning rate by multiplying it by a decay rate.

```
input_size = 32 * 32 * 3
hidden_size = 200
num_classes = 10
net = ThreeLayerNet(input_size, hidden_size, num_classes)

# Train the network
stats = net.train(X_train, y_train, X_val, y_val,
            num_iters=10000, batch_size=150,
            learning_rate=5e-3, learning_rate_decay=0.95,
            reg=0.01, verbose=True)

# Predict on the validation set
val_acc = (net.predict(X_val) == y_val).mean()
print('Validation accuracy: ', val_acc)
```

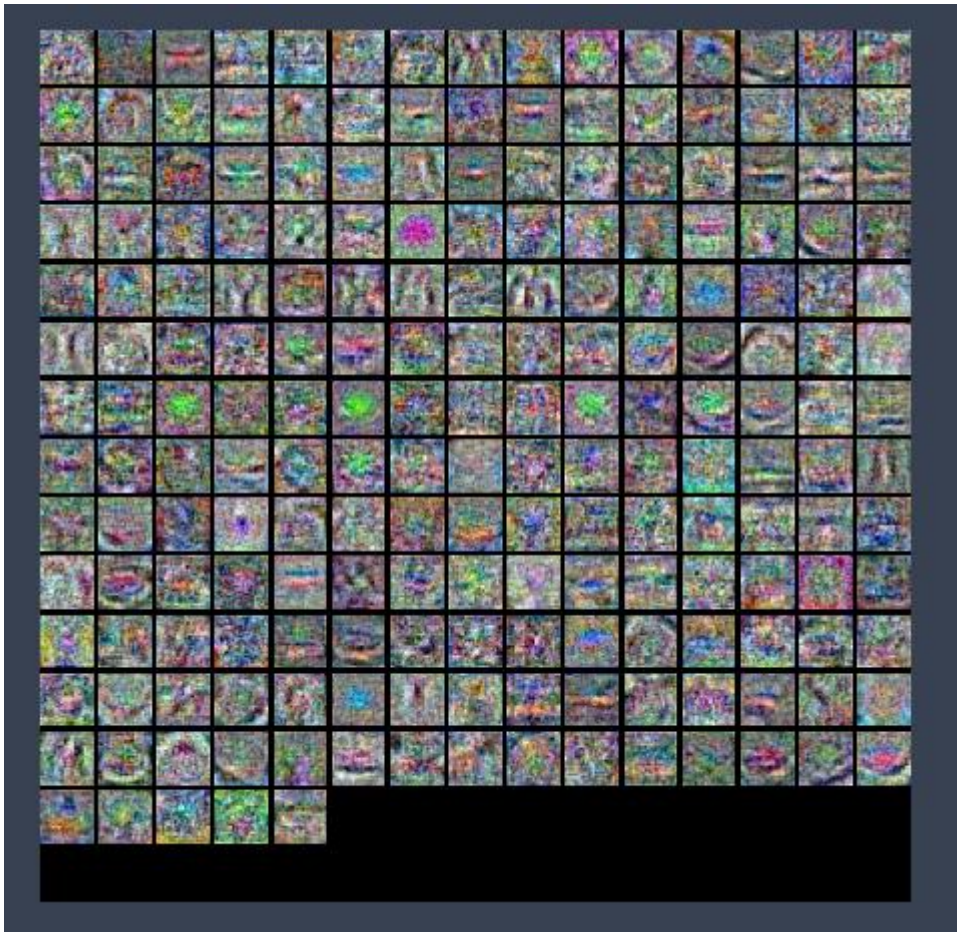
```
iteration 9400 / 10000: loss 1.041974
iteration 9500 / 10000: loss 0.923153
iteration 9600 / 10000: loss 0.968721
iteration 9700 / 10000: loss 1.078445
iteration 9800 / 10000: loss 0.937956
iteration 9900 / 10000: loss 1.095943
Validation accuracy:  0.56
```

经过修改超参后，测试结果达到 0.56

结果绘图：


Loss history
Loss
2.2
2.0
1.8
1.6
1.4
1.2
1.0
0.8
0
2000
4000
6000
8000
10000
Iteration
Classification accuracy history
Classification accuracy
0.140
0.135
0.130
0.125
0.120
0.115
train
val
−0.04
−0.02
0.00
0.02
0.04
Epoch

权重可视化：



搜索超参：

```
#pass
input_size = 32 * 32 * 3
num_classes = 10
best_acc = -1.0
hidden_size=200
# Train the network
for lr in [5e-3 ,1e-3, 1e-2]:
    for reg in [1e-4, 5e-4, 1e-3, 4e-3,1e-2,0.5,0.25]:
        net = ThreeLayerNet(input_size, hidden_size, num_classes)
        stats = net.train(X_train, y_train, X_val, y_val,
                    num_iters=20000, batch_size=150,
                    learning_rate=lr, learning_rate_decay=0.95,
                    reg=reg, verbose=True)

        # Predict on the validation set
        val_acc = (net.predict(X_val) == y_val).mean()
        print('reg = %f, lr = %f, Valid_accuracy: %f' %(reg, lr, val_a
        if val_acc > best_acc:
            best_acc = val_acc
            best_net = net

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

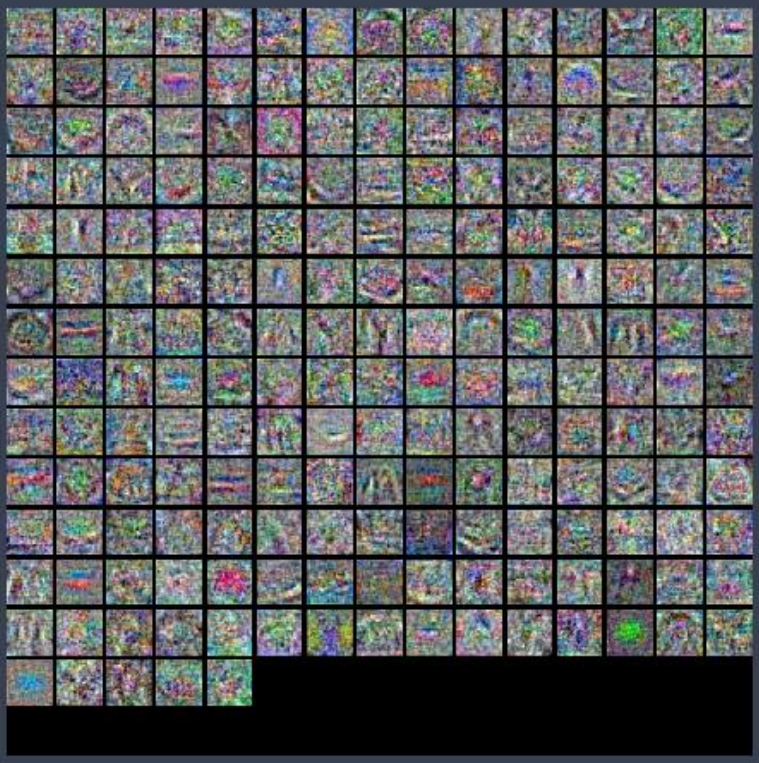
结果达到 0.542

```
iteration 19200 / 20000: loss 0.750496
iteration 19300 / 20000: loss 0.869285
iteration 19400 / 20000: loss 0.797552
iteration 19500 / 20000: loss 0.813169
iteration 19600 / 20000: loss 0.822161
iteration 19700 / 20000: loss 0.706738
iteration 19800 / 20000: loss 0.795891
iteration 19900 / 20000: loss 0.819784
reg = 0.010000, lr = 0.005000, Valid_accuracy: 0.542000
```

权重可视化：

```
# visualize the weights of the best network
show_net_weights(best_net)
```

executed in 162ms, finished 16:49:58 2021-10-05



10 Run on the test set

When you are done experimenting, you should evaluate your final trained network on the test set; you should get above 48%.

```
test_acc = (best_net.predict(X_test) == y_test).mean()
print('Test accuracy: ', test_acc)
```

executed in 39ms, finished 16:50:02 2021-10-05

```
Test accuracy:  0.537
```

Inline Question

Now that you have trained a Neural Network classifier, you may find that your testing accuracy is much lower than the training accuracy. In what ways can we decrease this gap? Select all that apply.

1. Train on a larger dataset.
2. Add more hidden units.
3. Increase the regularization strength.
4. None of the above.

Your Answer：*1,2,3* *Your Explanation*：*这个现象是过拟合的典型特征，可以通过增加训练集，增加网络参数数量，加大正则化减小过拟合。*

Features:

SVM:

```
#pass
for lr in learning_rates:
    for rs in regularization_strengths:
        svm = LinearSVM()
        loss_hist = svm.train(X_train_feats, y_train, learning_rate=lr
                    num_iters=1500, verbose=True)
        y_train_pred = svm.predict(X_train_feats)
        train_acc = np.mean(y_train == y_train_pred)
        y_val_pred = svm.predict(X_val_feats)
        val_acc = np.mean(y_val == y_val_pred)
        results[(lr, rs)] = (train_acc, val_acc)
        if val_acc > best_val:
            best_val = val_acc
            best_svm = svm

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

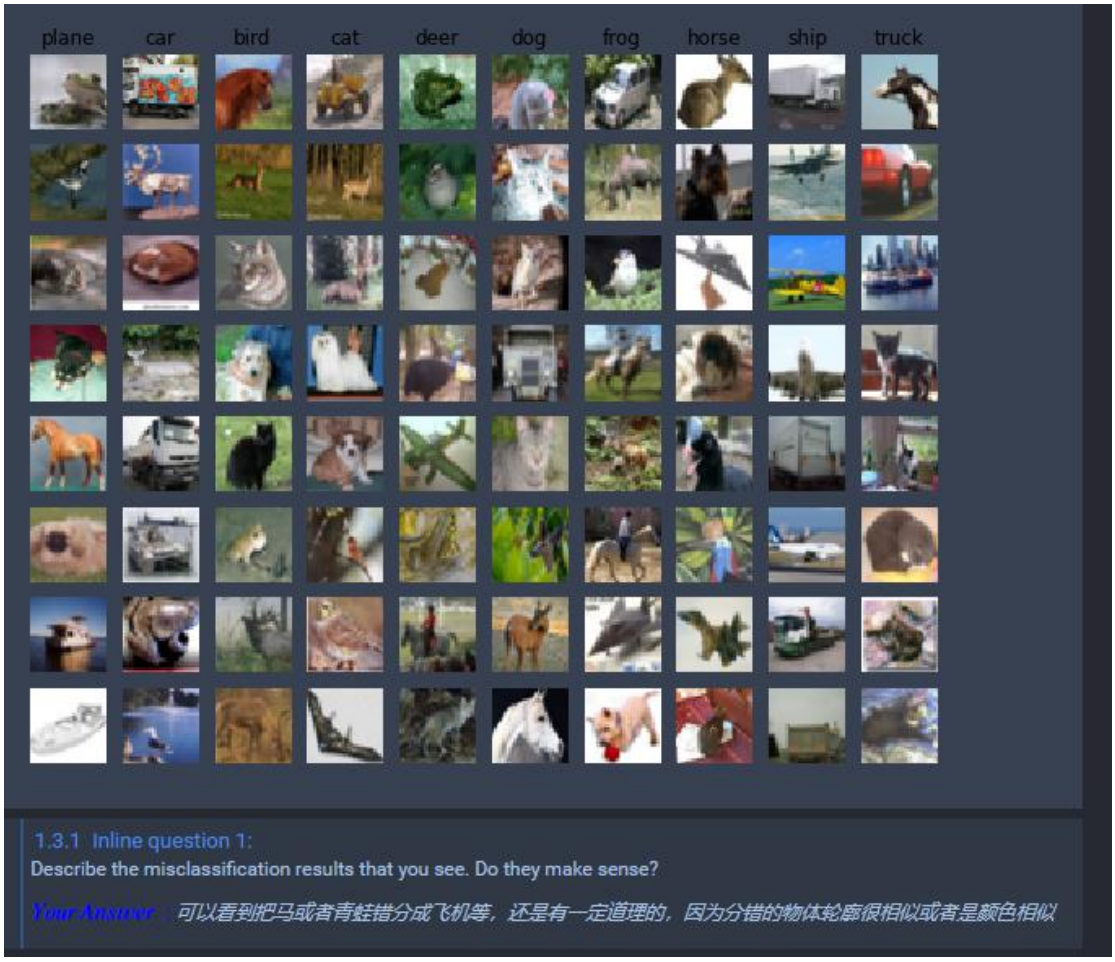
```
iteration 1000 / 1500: loss 8.999999
iteration 1100 / 1500: loss 8.999999
iteration 1200 / 1500: loss 9.000001
iteration 1300 / 1500: loss 9.000000
iteration 1400 / 1500: loss 9.000000
lr 1.000000e-09 reg 5.000000e+04 train accuracy: 0.102041 val accuracy: 0.116000
lr 1.000000e-09 reg 5.000000e+05 train accuracy: 0.107612 val accuracy: 0.113000
lr 1.000000e-09 reg 5.000000e+06 train accuracy: 0.413918 val accuracy: 0.418000
lr 1.000000e-08 reg 5.000000e+04 train accuracy: 0.130041 val accuracy: 0.147000
lr 1.000000e-08 reg 5.000000e+05 train accuracy: 0.413265 val accuracy: 0.412000
lr 1.000000e-08 reg 5.000000e+06 train accuracy: 0.415224 val accuracy: 0.410000
lr 1.000000e-07 reg 5.000000e+04 train accuracy: 0.416939 val accuracy: 0.416000
lr 1.000000e-07 reg 5.000000e+05 train accuracy: 0.402327 val accuracy: 0.406000
lr 1.000000e-07 reg 5.000000e+06 train accuracy: 0.324939 val accuracy: 0.308000
best validation accuracy achieved during cross-validation: 0.418000
```

```
# Evaluate your trained SVM on the test set
y_test_pred = best_svm.predict(X_test_feats)
test_accuracy = np.mean(y_test == y_test_pred)
print(test_accuracy)
```

executed in 14ms, finished 17:14:27 2021-10-06

```
0.415
```

分类结果



三层神经网络：

```
#pass
best_val = -1
result = {}

for learning_rate in [5e-1, 1.0,2.0]:
    for reg in [1e-6, 1e-5, 5e-5]:
        net = ThreeLayerNet(input_dim, hidden_dim, num_classes)
        stats=net.train(X_train_feats, y_train, X_val_feats, y_val,num
        train_acc = np.mean(y_train == net.predict(X_train_feats))
        val_acc = np.mean(y_val == net.predict(X_val_feats))
        result[(learning_rate, reg)] = train_acc, val_acc
        if  val_acc>best_val :
            best_val = val_acc
            best_net = net
        print('lr %e reg %e training acc: %f val acc: %f'%(lr,reg,trai

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

```
iteration 8800 / 10000: loss 0.726201
iteration 8900 / 10000: loss 0.816093
iteration 9000 / 10000: loss 0.745032
iteration 9100 / 10000: loss 0.719939
iteration 9200 / 10000: loss 0.603814
iteration 9300 / 10000: loss 0.670556
iteration 9400 / 10000: loss 0.585611
iteration 9500 / 10000: loss 0.610042
iteration 9600 / 10000: loss 0.586123
iteration 9700 / 10000: loss 0.798188
iteration 9800 / 10000: loss 0.697355
iteration 9900 / 10000: loss 0.673101
lr 1.000000e-07 reg 5.000000e-05 training acc: 0.826041 val acc: 0.579000
```

可以看到 val 准确度达到 0.579

```
# Run your best neural net classifier on the test set. You should be a
# to get more than 55% accuracy.

test_acc = (best_net.predict(X_test_feats) == y_test).mean()
print(test_acc)
```

```
executed in 50ms, finished 17:50:02 2021-10-06
```

```
0.577
```

测试集准确度达到 0.577

结论分析与体会：

1，要时刻注意正则化，对参数进行惩罚最主要的作用其实是防止过拟合（overfitting），提高模型的泛化能力。
2，偏置 b 不会对输入特征的影响强度产生作用，所以我们不需要对 b 进行惩罚（但是 b 被合并到了 W 里，所以实际上我们对 b 也进行了惩罚，不过影响不大）。
3，调整超参的时候使用最简单的网格搜索，需要多指定范围，可以先把 batch_size 一开始调的很大，加快训练速度，然后粗调得到较好结果后再调小，网络到三层的时候需要相应调整隐藏层的大小，学习率可以一开始调大，然后使用衰减，正则化系数可以往小了调。
4，在计算反向传播的时候，需要弄清楚误差反向传播的过程，然后一步一步，一层一层的向前传，注意最后一层没有激活函数。
5，画出权重图后看到分类效果并不是很好，而且权重图可能很模糊，只是一个大概的轮廓或者颜色。
6，在分析模型结果的时候，不仅要考虑模型本身的效果，还要考虑数据的噪声或者分布不均等。
7，在观察分类结果对比我们发现神经网络会比 knn、svm 等模型效果要好，不过神经网络的精确度需要不停的调整超参，同时要保证训练过程的正确，这里我们使用 SGD 作为优化方案，所以要清楚 SGD 的运算过程。