k-Nearest Neighbor (kNN) exercise

Complete and hand in this completed worksheet (including its outputs and any supporting code outside of the worksheet) with your assignment submission. For more details see the [assignments page (https://compsci682-fa19.github.io/assignments2019/assignment1)](https://compsci682-fa19.github.io/assignments2019/assignment1) on the course website.

The kNN classifier consists of two stages:

- During training, the classifier takes the training data and simply remembers it
- During testing, kNN classifies every test image by comparing to all training images and transfering the labels of the k most similar training examples
- The value of k is cross-validated

In this exercise you will implement these steps and understand the basic Image Classification pipeline, cross-validation, and gain proficiency in writing efficient, vectorized code.

In [121]:

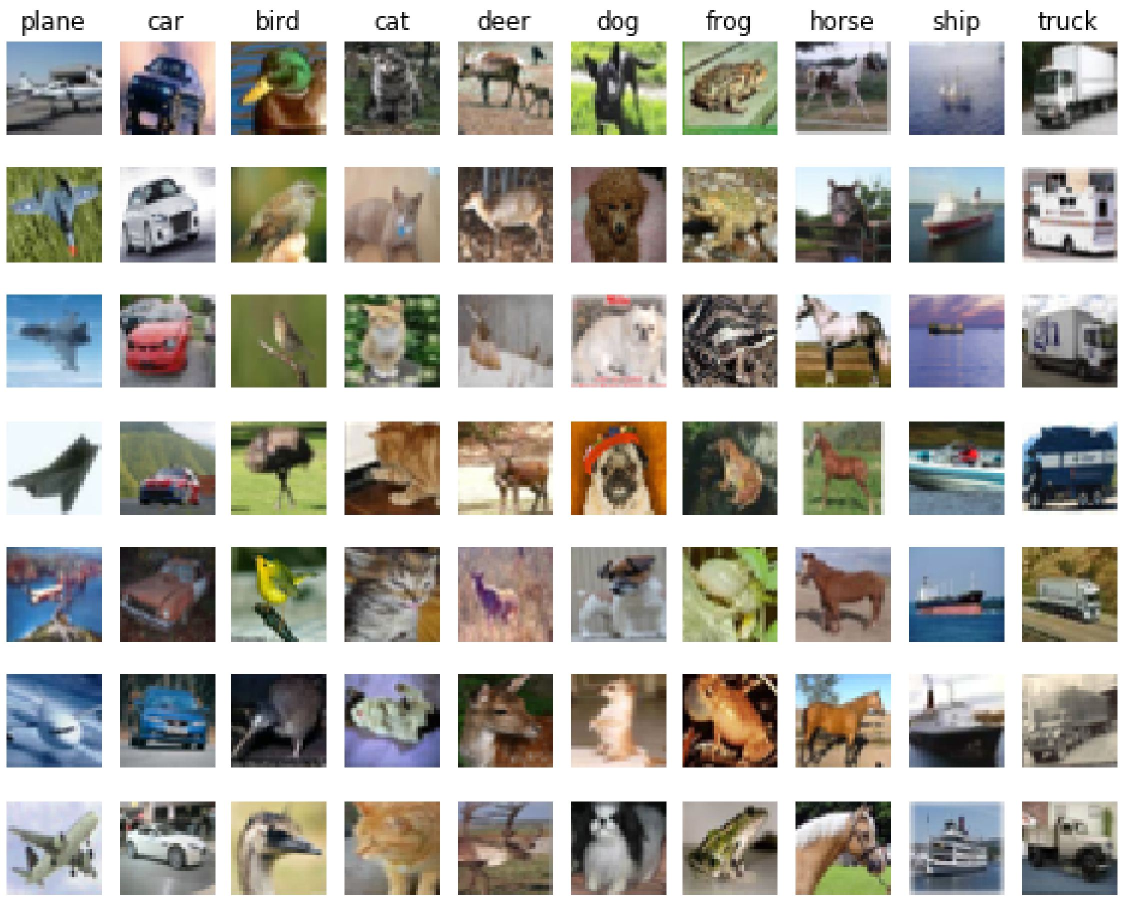
```
# Run some setup code for this notebook.
from __future__ import print_function

import random
import numpy as np
from cs682.data_utils import load_CIFAR10
import matplotlib.pyplot as plt


# This is a bit of magic to make matplotlib figures appear inline in
 the notebook
# rather than in a new window.
%matplotlib inline
plt.rcParams['figure.figsize'] = (10.0, 8.0) # set default size of pl
ots
plt.rcParams['image.interpolation'] = 'nearest'
plt.rcParams['image.cmap'] = 'gray'

# Some more magic so that the notebook will reload external python mo
dules;
# see http://stackoverflow.com/questions/1907993/autoreload-of-module
s-in-ipython
%load_ext autoreload
%autoreload 2
```

```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
```

In [122]:
```
# Load the raw CIFAR-10 data.
cifar10_dir = 'cs682/datasets/cifar-10-batches-py'

# Cleaning up variables to prevent loading data multiple times (which
may cause memory issue)
try:
   del X_train, y_train
   del X_test, y_test
   print('Clear previously loaded data.')
except:
   pass

X_train, y_train, X_test, y_test = load_CIFAR10(cifar10_dir)

# As a sanity check, we print out the size of the training and test d
ata.
print('Training data shape: ', X_train.shape)
print('Training labels shape: ', y_train.shape)
print('Test data shape: ', X_test.shape)
print('Test labels shape: ', y_test.shape)
```

```
Clear previously loaded data.
Training data shape:  (50000, 32, 32, 3)
Training labels shape:  (50000,)
Test data shape:  (10000, 32, 32, 3)
Test labels shape:  (10000,)
```

In [111]:
```
# Visualize some examples from the dataset.
# We show a few examples of training images from each class.
classes = ['plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'hor
se', 'ship', 'truck']
num_classes = len(classes)
samples_per_class = 7
for y, cls in enumerate(classes):
    idxs = np.flatnonzero(y_train == y)
    idxs = np.random.choice(idxs, samples_per_class, replace=False)
    for i, idx in enumerate(idxs):
        plt_idx = i * num_classes + y + 1
        plt.subplot(samples_per_class, num_classes, plt_idx)
        plt.imshow(X_train[idx].astype('uint8'))
        plt.axis('off')
        if i == 0:
            plt.title(cls)
plt.show()
```



In [123]:
```
# Subsample the data for more efficient code execution in this exerci
se
num_training = 5000
mask = list(range(num_training))
X_train = X_train[mask]
y_train = y_train[mask]

num_test = 500
mask = list(range(num_test))
X_test = X_test[mask]
y_test = y_test[mask]
```

In [124]:
```
# Reshape the image data into rows
X_train = np.reshape(X_train, (X_train.shape[0], -1))
X_test = np.reshape(X_test, (X_test.shape[0], -1))
print(X_train.shape, X_test.shape)
```

```
(5000, 3072) (500, 3072)
```

In [125]:
```
from cs682.classifiers import KNearestNeighbor

# Create a kNN classifier instance.
# Remember that training a kNN classifier is a noop:
# the Classifier simply remembers the data and does no further proces
sing
classifier = KNearestNeighbor()
classifier.train(X_train, y_train)
```

We would now like to classify the test data with the kNN classifier. Recall that we can break down this process into two steps:

1. First we must compute the distances between all test examples and all train examples.
2. Given these distances, for each test example we find the k nearest examples and have them vote for the label

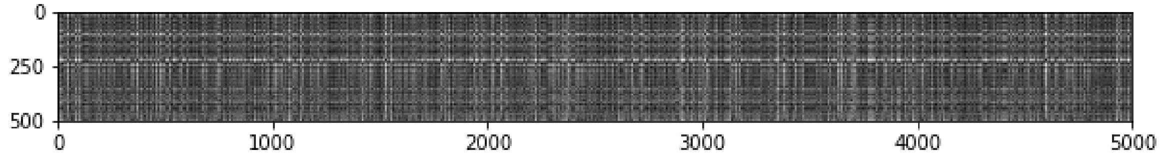
Lets begin with computing the distance matrix between all training and test examples. For example, if there are **Ntr** training examples and **Nte** test examples, this stage should result in a **Nte x Ntr** matrix where each element (i,j) is the distance between the i-th test and j-th train example.

First, open `cs682/classifiers/k_nearest_neighbor.py` and implement the function `compute_distances_two_loops` that uses a (very inefficient) double loop over all pairs of (test, train) examples and computes the distance matrix one element at a time.

In [126]:
```
# Open cs682/classifiers/k_nearest_neighbor.py and implement
# compute_distances_two_loops.

# Test your implementation:
dists = classifier.compute_distances_two_loops(X_test)
print(dists.shape)
```

```
(500, 5000)
```

In [115]:
```
# We can visualize the distance matrix: each row is a single test exa
mple and
# its distances to training examples
plt.imshow(dists, interpolation='none')
plt.show()
```



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. Distinctly bright rows means that their distance from other points is huge. It is possible that certain points in testing set are really far from the training data points, which is why their distance is very high. These images must be very different from most of the training set, making them outliers.
2. Bright columns are caused by images which are outliers in training set.

In [127]:
```
# Now implement the function predict_labels and run the code below:
# We use k = 1 (which is Nearest Neighbor).
y_test_pred = classifier.predict_labels(dists, k=1)

# Compute and print the fraction of correctly predicted examples
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))
```

```
Got 137 / 500 correct => accuracy: 0.274000
```

You should expect to see approximately `27%` accuracy. Now lets try out a larger `k`, say `k = 5`:

In [128]:
```
y_test_pred = classifier.predict_labels(dists, k=5)
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))
```

```
Got 139 / 500 correct => accuracy: 0.278000
```

You should expect to see a slightly better performance than with `k = 1`.

Inline Question 2 We can also other distance metrics such as L1 distance. The performance of a Nearest Neighbor classifier that uses L1 distance will not change if (Select all that apply.):

1. The data is preprocessed by subtracting the mean.
2. The data is preprocessed by subtracting the mean and dividing by the standard deviation.
3. The coordinate axes for the data are rotated.
4. None of the above.

Your Answer: 1,2 - TRUE

Your explanation: Boundaries for nearest neighbour classifer (NNC) that uses L1 distance are in the shape of a rotated square.

1. When any constant is removed from all the points in the data, their relative positions remain unchanged, hence L1 distance between them remain unchanged. hence the performance remains the same.
2. When constant is removed and divided by a single number, their L1 distance is compressed by that factor but their relative distance remain unchanged. Hence, the performance remains the same.
3. When axes are totated, L1 distance between the points changes unequally, resulting in a different performance. So, this is not true.

In [129]:
```
# Now lets speed up distance matrix computation by using partial vect
orization
# with one loop. Implement the function compute_distances_one_loop an
d run the
# code below:
dists_one = classifier.compute_distances_one_loop(X_test)

# To ensure that our vectorized implementation is correct, we make su
re that it
# agrees with the naive implementation. There are many ways to decide
whether
# two matrices are similar; one of the simplest is the Frobenius nor
m. In case
# you haven't seen it before, the Frobenius norm of two matrices is t
he square
# root of the squared sum of differences of all elements; in other wo
rds, reshape
# the matrices into vectors and compute the Euclidean distance betwee
n them.
difference = np.linalg.norm(dists - dists_one, ord='fro')
print('Difference was: %f' % (difference, ))
if difference < 0.001:
    print('Good! The distance matrices are the same')
else:
    print('Uh-oh! The distance matrices are different')
```

```
Difference was: 0.000000
Good! The distance matrices are the same
```

In [130]:
```
# Now implement the fully vectorized version inside compute_distances
_no_loops
# and run the code
dists_two = classifier.compute_distances_no_loops(X_test)

# check that the distance matrix agrees with the one we computed befo
re:
difference = np.linalg.norm(dists - dists_two, ord='fro')
print('Difference was: %f' % (difference, ))
if difference < 0.001:
    print('Good! The distance matrices are the same')
else:
    print('Uh-oh! The distance matrices are different')
```

```
Difference was: 0.000000
Good! The distance matrices are the same
```

In [131]:
```
# Let's compare how fast the implementations are
def time_function(f, *args):
    """
    Call a function f with args and return the time (in seconds) that
it took to execute.
    """
    import time
    tic = time.time()
    f(*args)
    toc = time.time()
    return toc - tic

two_loop_time = time_function(classifier.compute_distances_two_loops,
X_test)
print('Two loop version took %f seconds' % two_loop_time)

one_loop_time = time_function(classifier.compute_distances_one_loop,
X_test)
print('One loop version took %f seconds' % one_loop_time)

no_loop_time = time_function(classifier.compute_distances_no_loops, X
_test)
print('No loop version took %f seconds' % no_loop_time)

# you should see significantly faster performance with the fully vect
orized implementation
```

```
Two loop version took 17.916950 seconds
One loop version took 42.243923 seconds
No loop version took 0.161431 seconds
```

Cross-validation

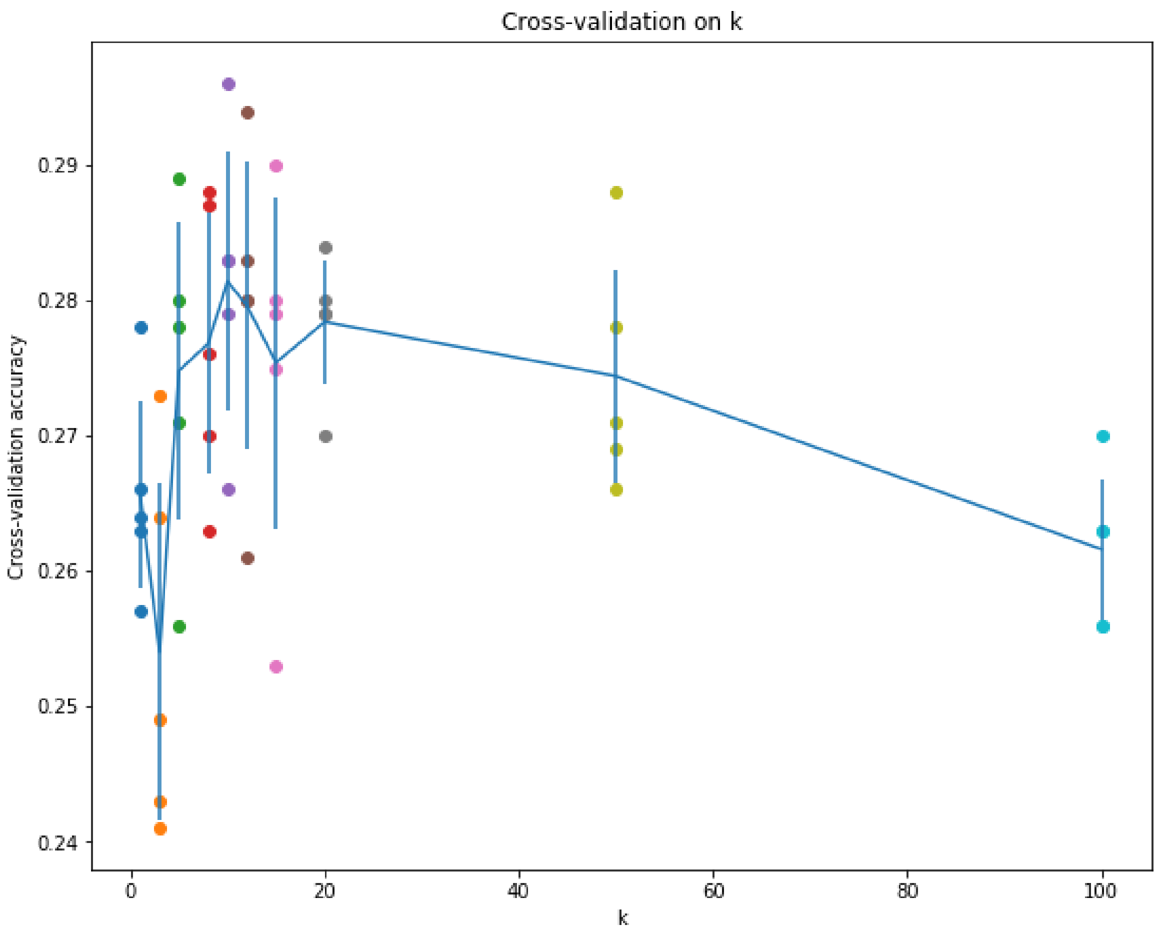
We have implemented the k-Nearest Neighbor classifier but we set the value k = 5 arbitrarily. We will now determine the best value of this hyperparameter with cross-validation.

In [132]:
```
num_folds = 5
k_choices = [1, 3, 5, 8, 10, 12, 15, 20, 50, 100]

X_train_folds = []
y_train_folds = []
#####################################################################
###########
# TODO:
#
# Split up the training data into folds. After splitting, X_train_fol
ds and    #
# y_train_folds should each be lists of length num_folds, where
#
# y_train_folds[i] is the label vector for the points in X_train_fold
s[i].     #
# Hint: Look up the numpy array_split function.
#
#####################################################################
###########
# Your code
X_train_folds = np.split(X_train, num_folds)
y_train_folds = np.split(y_train, num_folds)

#####################################################################
###########
#                                  END OF YOUR CODE
#
#####################################################################
###########

# A dictionary holding the accuracies for different values of k that
 we find
# when running cross-validation. After running cross-validation,
# k_to_accuracies[k] should be a list of length num_folds giving the
 different
# accuracy values that we found when using that value of k.
k_to_accuracies = {}


#####################################################################
###########
# TODO:
#
# Perform k-fold cross validation to find the best value of k. For ea
ch        #
# possible value of k, run the k-nearest-neighbor algorithm num_folds
times,   #
# where in each case you use all but one of the folds as training dat
a and the #
# last fold as a validation set. Store the accuracies for all fold an
d all     #
# values of k in the k_to_accuracies dictionary.
#
#####################################################################
###########
# Your code
```

```
for k in k_choices:
    classifier = KNearestNeighbor()
    accuracies = []
    for fold in np.arange(num_folds):
        X_train_current = np.zeros((0,X_train_folds[0].shape[1]), dty
pe = int)
        y_train_current = np.zeros((0,), dtype = int)
        for i in np.arange(num_folds):
            if i != fold:
                X_train_current = np.concatenate((X_train_current, X_
train_folds[i]), axis = 0)
                y_train_current = np.concatenate((y_train_current, y_
train_folds[i]), axis = 0)


        classifier.train(X_train_current, y_train_current)
        y_test_pred = classifier.predict(X_train_folds[fold], k=k)
        num_correct = np.sum(y_test_pred == y_train_folds[fold])
        accuracy = float(num_correct) / y_train_folds[fold].shape[0]
        accuracies.append(accuracy)

    k_to_accuracies[k] = accuracies
####################################################################
###########
#                                 END OF YOUR CODE
#
####################################################################
###########

# Print out the computed accuracies
for k in sorted(k_to_accuracies):
    for accuracy in k_to_accuracies[k]:
        print('k = %d, accuracy = %f' % (k, accuracy))
```

```
k = 1, accuracy = 0.263000
k = 1, accuracy = 0.257000
k = 1, accuracy = 0.264000
k = 1, accuracy = 0.278000
k = 1, accuracy = 0.266000
k = 3, accuracy = 0.241000
k = 3, accuracy = 0.249000
k = 3, accuracy = 0.243000
k = 3, accuracy = 0.273000
k = 3, accuracy = 0.264000
k = 5, accuracy = 0.256000
k = 5, accuracy = 0.271000
k = 5, accuracy = 0.280000
k = 5, accuracy = 0.289000
k = 5, accuracy = 0.278000
k = 8, accuracy = 0.263000
k = 8, accuracy = 0.287000
k = 8, accuracy = 0.276000
k = 8, accuracy = 0.288000
k = 8, accuracy = 0.270000
k = 10, accuracy = 0.266000
k = 10, accuracy = 0.296000
k = 10, accuracy = 0.279000
k = 10, accuracy = 0.283000
k = 10, accuracy = 0.283000
k = 12, accuracy = 0.261000
k = 12, accuracy = 0.294000
k = 12, accuracy = 0.280000
k = 12, accuracy = 0.283000
k = 12, accuracy = 0.280000
k = 15, accuracy = 0.253000
k = 15, accuracy = 0.290000
k = 15, accuracy = 0.279000
k = 15, accuracy = 0.280000
k = 15, accuracy = 0.275000
k = 20, accuracy = 0.270000
k = 20, accuracy = 0.279000
k = 20, accuracy = 0.279000
k = 20, accuracy = 0.280000
k = 20, accuracy = 0.284000
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
```

In [133]:
```
# plot the raw observations
for k in k_choices:
    accuracies = k_to_accuracies[k]
    plt.scatter([k] * len(accuracies), accuracies)

# plot the trend line with error bars that correspond to standard dev
iation
accuracies_mean = np.array([np.mean(v) for k,v in sorted(k_to_accurac
ies.items())])
accuracies_std = np.array([np.std(v) for k,v in sorted(k_to_accuracie
s.items())])
plt.errorbar(k_choices, accuracies_mean, yerr=accuracies_std)
plt.title('Cross-validation on k')
plt.xlabel('k')
plt.ylabel('Cross-validation accuracy')
plt.show()
```



In [134]:
```
# Based on the cross-validation results above, choose the best value
 for k,
# retrain the classifier using all the training data, and test it on
 the test
# data. You should be able to get above 28% accuracy on the test dat
a.
best_k = 10

classifier = KNearestNeighbor()
classifier.train(X_train, y_train)
y_test_pred = classifier.predict(X_test, k=best_k)

# 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))
```

```
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 training error of a 1-NN will always be better than that of 5-NN.
2. The test error of a 1-NN will always be better than that of a 5-NN.
3. The decision boundary of the k-NN classifier is linear.
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: 1,4 - TRUE

Your explanation:

1. Training error of 1-NN is zero since we are seeing the difference between the same set of points. This will always be better than or equal to that of 5-NN. (Equality in very special case, where all the points of each class are closer to themselves than the other class). So, this is TRUE.
2. Test error of 1-NN can be better or worse than 5-NN depending on the case. So, this is FALSE.
3. Boundary is not linear. Consider three concentric circles of points. Here, the boundaries would most probably be circular. Hence, FALSE.
4. Time need to classify a test example is the time needed to evaluate the distance of the test example from each of the training example and taking the least k-distances. This computation time increases with the number of training examples. So, this is TRUE.

Multiclass Support Vector Machine exercise

Complete and hand in this completed worksheet (including its outputs and any supporting code outside of the worksheet) with your assignment submission. For more details see the [assignments page (https://compsci682-fa19.github.io/assignments2019/assignment1/)](https://compsci682-fa19.github.io/assignments2019/assignment1/) on the course website.

In this exercise you will:

- implement a fully-vectorized **loss function** for the SVM
- implement the fully-vectorized expression for its **analytic gradient**
- **check your implementation** using numerical gradient
- use a validation set to **tune the learning rate and regularization** strength
- **optimize** the loss function with **SGD**
- **visualize** the final learned weights

In [24]:
```
# Run some setup code for this notebook.
from __future__ import print_function
import random
import numpy as np
from cs682.data_utils import load_CIFAR10
import matplotlib.pyplot as plt


# This is a bit of magic to make matplotlib figures appear inline in
 the
# notebook rather than in a new window.
%matplotlib inline
plt.rcParams['figure.figsize'] = (10.0, 8.0) # set default size of pl
ots
plt.rcParams['image.interpolation'] = 'nearest'
plt.rcParams['image.cmap'] = 'gray'

# Some more magic so that the notebook will reload external python mo
dules;
# see http://stackoverflow.com/questions/1907993/autoreload-of-module
s-in-ipython
%load_ext autoreload
%autoreload 2
```

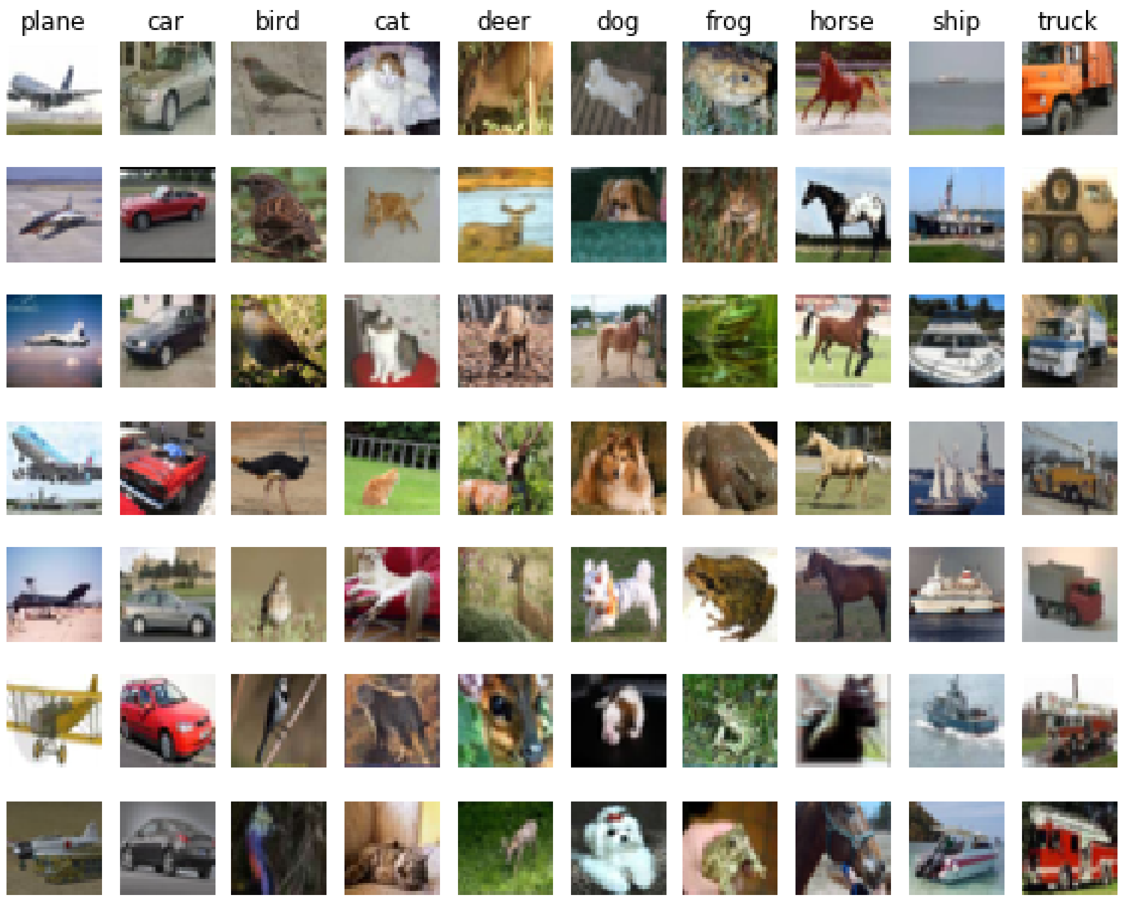
```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
```

CIFAR-10 Data Loading and Preprocessing

In [25]:
```
# Load the raw CIFAR-10 data.
cifar10_dir = 'cs682/datasets/cifar-10-batches-py'

# Cleaning up variables to prevent loading data multiple times (which
may cause memory issue)
try:
   del X_train, y_train
   del X_test, y_test
   print('Clear previously loaded data.')
except:
   pass

X_train, y_train, X_test, y_test = load_CIFAR10(cifar10_dir)

# As a sanity check, we print out the size of the training and test d
ata.
print('Training data shape: ', X_train.shape)
print('Training labels shape: ', y_train.shape)
print('Test data shape: ', X_test.shape)
print('Test labels shape: ', y_test.shape)
```

```
Clear previously loaded data.
Training data shape:  (50000, 32, 32, 3)
Training labels shape:  (50000,)
Test data shape:  (10000, 32, 32, 3)
Test labels shape:  (10000,)
```

In [26]:
```
# Visualize some examples from the dataset.
# We show a few examples of training images from each class.
classes = ['plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'hor
se', 'ship', 'truck']
num_classes = len(classes)
samples_per_class = 7
for y, cls in enumerate(classes):
    idxs = np.flatnonzero(y_train == y)
    idxs = np.random.choice(idxs, samples_per_class, replace=False)
    for i, idx in enumerate(idxs):
        plt_idx = i * num_classes + y + 1
        plt.subplot(samples_per_class, num_classes, plt_idx)
        plt.imshow(X_train[idx].astype('uint8'))
        plt.axis('off')
        if i == 0:
            plt.title(cls)
plt.show()
```



In [27]:

```
# Split the data into train, val, and test sets. In addition we will
# create a small development set as a subset of the training data;
# we can use this for development so our code runs faster.
num_training = 49000
num_validation = 1000
num_test = 1000
num_dev = 500

# Our validation set will be num_validation points from the original
# training set.
mask = range(num_training, num_training + num_validation)
X_val = X_train[mask]
y_val = y_train[mask]

# Our training set will be the first num_train points from the origin
al
# training set.
mask = range(num_training)
X_train = X_train[mask]
y_train = y_train[mask]

# We will also make a development set, which is a small subset of
# the training set.
mask = np.random.choice(num_training, num_dev, replace=False)
X_dev = X_train[mask]
y_dev = y_train[mask]

# We use the first num_test points of the original test set as our
# test set.
mask = range(num_test)
X_test = X_test[mask]
y_test = y_test[mask]

print('Train data shape: ', X_train.shape)
print('Train labels shape: ', y_train.shape)
print('Validation data shape: ', X_val.shape)
print('Validation labels shape: ', y_val.shape)
print('Test data shape: ', X_test.shape)
print('Test labels shape: ', y_test.shape)
```

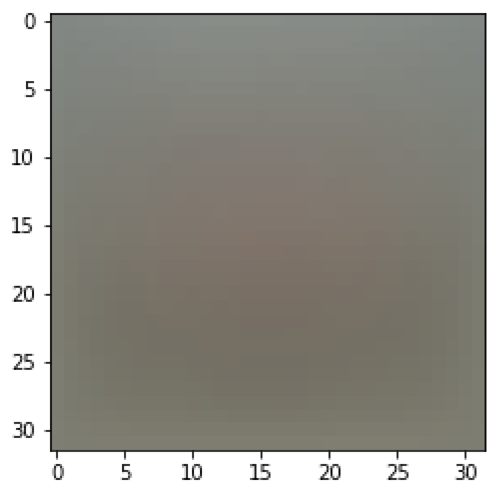
```
Train data shape:  (49000, 32, 32, 3)
Train labels shape:  (49000,)
Validation data shape:  (1000, 32, 32, 3)
Validation labels shape:  (1000,)
Test data shape:  (1000, 32, 32, 3)
Test labels shape:  (1000,)
```

In [28]:
```
# Preprocessing: reshape the image data into rows
X_train = np.reshape(X_train, (X_train.shape[0], -1))
X_val = np.reshape(X_val, (X_val.shape[0], -1))
X_test = np.reshape(X_test, (X_test.shape[0], -1))
X_dev = np.reshape(X_dev, (X_dev.shape[0], -1))

# As a sanity check, print out the shapes of the data
print('Training data shape: ', X_train.shape)
print('Validation data shape: ', X_val.shape)
print('Test data shape: ', X_test.shape)
print('dev data shape: ', X_dev.shape)
```

```
Training data shape:  (49000, 3072)
Validation data shape:  (1000, 3072)
Test data shape:  (1000, 3072)
dev data shape:  (500, 3072)
```

In [29]:
```
# Preprocessing: subtract the mean image
# first: compute the image mean based on the training data
mean_image = np.mean(X_train, axis=0)
print(mean_image[:10]) # print a few of the elements
plt.figure(figsize=(4,4))
plt.imshow(mean_image.reshape((32,32,3)).astype('uint8')) # visualize the mean image
plt.show()
```

```
[130.64189796 135.98173469 132.47391837 130.05569388 135.34804082
 131.75402041 130.96055102 136.14328571 132.47636735 131.48467347]
```



In [30]:
```
# second: subtract the mean image from train and test data
X_train -= mean_image
X_val -= mean_image
X_test -= mean_image
X_dev -= mean_image
```

In [31]:
```
# third: append the bias dimension of ones (i.e. bias trick) so that
 our SVM
# only has to worry about optimizing a single weight matrix W.
X_train = np.hstack([X_train, np.ones((X_train.shape[0], 1))])
X_val = np.hstack([X_val, np.ones((X_val.shape[0], 1))])
X_test = np.hstack([X_test, np.ones((X_test.shape[0], 1))])
X_dev = np.hstack([X_dev, np.ones((X_dev.shape[0], 1))])

print(X_train.shape, X_val.shape, X_test.shape, X_dev.shape)
```

```
(49000, 3073) (1000, 3073) (1000, 3073) (500, 3073)
```

SVM Classifier

Your code for this section will all be written inside **cs682/classifiers/linear_svm.py**.

As you can see, we have prefilled the function `svm_loss_naive` which uses for loops to evaluate the multiclass SVM loss function.

In [32]:
```
# Evaluate the naive implementation of the loss we provided for you:
from cs682.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))
```

```
loss: 8.589740
```

The `grad` returned from the function above is right now all zero. Derive and implement the gradient for the SVM cost function and implement it inline inside the function `svm_loss_naive` . You will find it helpful to interleave your new code inside the existing function.

To check that you have correctly implemented the gradient correctly, you can numerically estimate the gradient of the loss function and compare the numeric estimate to the gradient that you computed. We have provided code that does this for you:

In [33]:
```
# Once you've implemented the gradient, recompute it with the code be
low
# and gradient check it with the function we provided for you

# Compute the loss adWnd its gradient at W.
loss, grad = svm_loss_naive(W, X_dev, y_dev, 0.0)

# Numerically compute the gradient along several randomly chosen dime
nsions, and
# compare them with your analytically computed gradient. The numbers
 should match
# almost exactly along all dimensions.
from cs682.gradient_check import grad_check_sparse
f = lambda w: svm_loss_naive(w, X_dev, y_dev, 0.0)[0]
grad_numerical = grad_check_sparse(f, W, grad)

# do the gradient check once again with regularization turned on
# you didn't forget the regularization gradient did you?
loss, grad = svm_loss_naive(W, X_dev, y_dev, 5e1)
f = lambda w: svm_loss_naive(w, X_dev, y_dev, 5e1)[0]
grad_numerical = grad_check_sparse(f, W, grad)
```

```
numerical: 27.485750 analytic: 27.485750, relative error: 1.388409e-1
2
numerical: 26.531183 analytic: 26.531183, relative error: 1.209201e-1
1
numerical: 9.232358 analytic: 9.232358, relative error: 6.625100e-12
numerical: 14.437954 analytic: 14.437954, relative error: 5.025255e-1
2
numerical: 22.083533 analytic: 22.083533, relative error: 1.631244e-1
1
numerical: -24.383923 analytic: -24.383923, relative error: 1.403322e
-11
numerical: 9.799457 analytic: 9.799457, relative error: 1.767147e-11
numerical: 14.759554 analytic: 14.759554, relative error: 6.750962e-1
2
numerical: 28.077529 analytic: 28.077529, relative error: 2.172559e-1
3
numerical: -1.233368 analytic: -1.233368, relative error: 6.199210e-1
1
numerical: 14.927535 analytic: 14.927535, relative error: 1.604054e-1
1
numerical: 2.365012 analytic: 2.365012, relative error: 3.384741e-11
numerical: -2.073142 analytic: -2.073142, relative error: 5.095891e-1
1
numerical: 7.855398 analytic: 7.855398, relative error: 3.405390e-11
numerical: -28.289928 analytic: -28.289928, relative error: 8.268965e
-13
numerical: 2.439470 analytic: 2.439470, relative error: 1.376263e-10
numerical: 23.936934 analytic: 23.936934, relative error: 7.175204e-1
2
numerical: -14.462851 analytic: -14.462851, relative error: 2.068563e
-11
numerical: 27.549003 analytic: 27.549003, relative error: 7.266884e-1
3
numerical: -5.732948 analytic: -5.732948, relative error: 5.745093e-1
2
```

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: The loss function will not be differentiable at the point where the loss is zero, since it is max(loss,0). This is not a cause of concern as it happens rarely. For example, consider the one-dimenstional example of X = 1. Assume W=(0,-1). Now, loss is max(-x + 0 + 1, 0). For little margins above and below X = 1, we will have very different gradients, i.e. 0 and -1 respectively. However, numerial computation will have very little difference at these points. This is the reason grad check not matching exactly once in a while.

In [34]:
```
# Next implement the function svm_loss_vectorized; for now only compu
te the loss;
# we will implement the gradient in a moment.
tic = time.time()
loss_naive, grad_naive = svm_loss_naive(W, X_dev, y_dev, 0.000005)
toc = time.time()
print('Naive loss: %e computed in %fs' % (loss_naive, toc - tic))

from cs682.classifiers.linear_svm import svm_loss_vectorized
tic = time.time()
loss_vectorized, _ = svm_loss_vectorized(W, X_dev, y_dev, 0.000005)
toc = time.time()
print('Vectorized loss: %e computed in %fs' % (loss_vectorized, toc -
tic))

# The losses should match but your vectorized implementation should b
e much faster.
print('difference: %f' % (loss_naive - loss_vectorized))
```

```
Naive loss: 8.589740e+00 computed in 0.066196s
Vectorized loss: 8.589740e+00 computed in 0.002009s
difference: 0.000000
```

In [35]:
```
# Complete the implementation of svm_loss_vectorized, and compute the
gradient
# of the loss function in a vectorized way.

# The naive implementation and the vectorized implementation should m
atch, but
# the vectorized version should still be much faster.
tic = time.time()
_, grad_naive = svm_loss_naive(W, X_dev, y_dev, 0.000005)
toc = time.time()
print('Naive loss and gradient: computed in %fs' % (toc - tic))

tic = time.time()
_, grad_vectorized = svm_loss_vectorized(W, X_dev, y_dev, 0.000005)
toc = time.time()
print('Vectorized loss and gradient: computed in %fs' % (toc - tic))

# The loss is a single number, so it is easy to compare the values co
mputed
# by the two implementations. The gradient on the other hand is a mat
rix, so
# we use the Frobenius norm to compare them.
difference = np.linalg.norm(grad_naive - grad_vectorized, ord='fro')
print('difference: %f' % difference)
```

```
Naive loss and gradient: computed in 0.065558s
Vectorized loss and gradient: computed in 0.002451s
difference: 0.000000
```

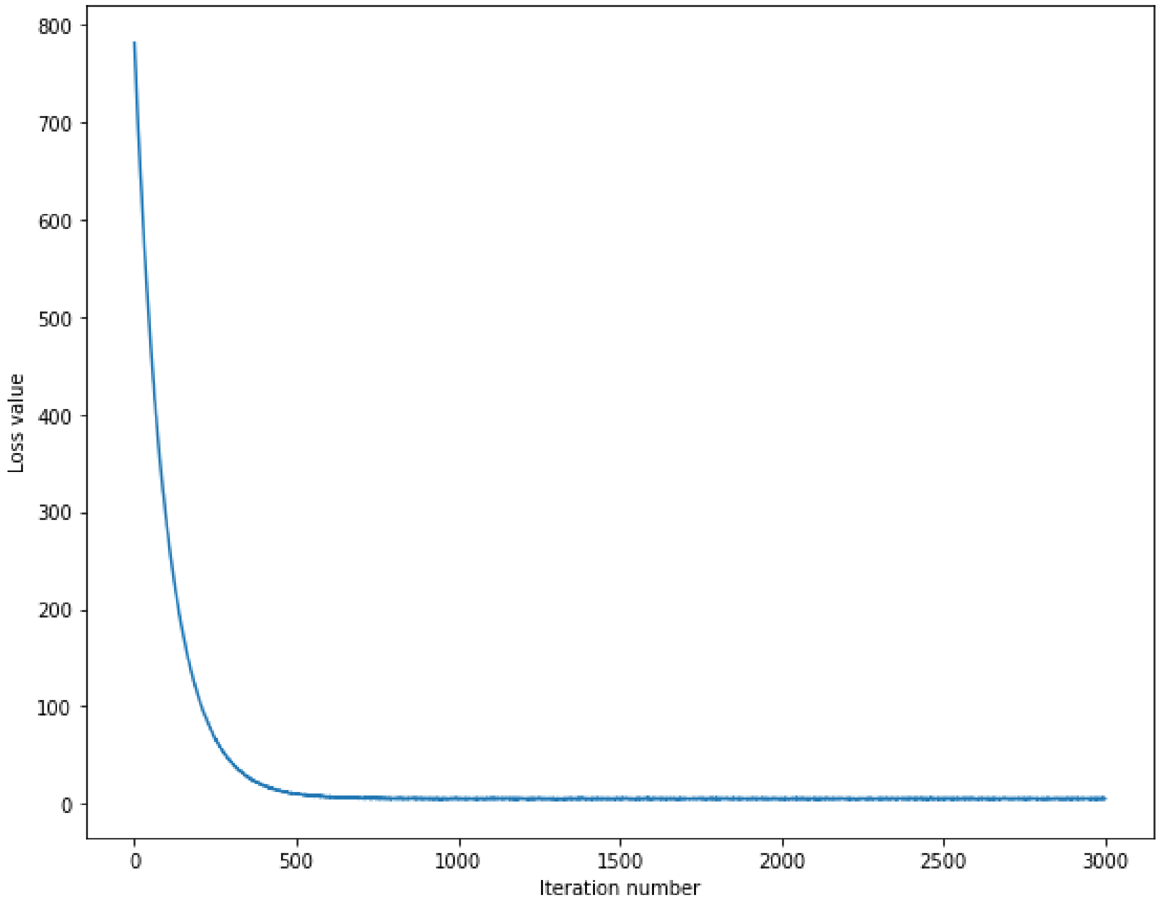
Stochastic Gradient Descent

We now have vectorized and efficient expressions for the loss, the gradient and our gradient matches the numerical gradient. We are therefore ready to do SGD to minimize the loss.

In [36]:
```
# In the file linear_classifier.py, implement SGD in the function
# LinearClassifier.train() and then run it with the code below.
from cs682.classifiers import LinearSVM
svm = LinearSVM()
tic = time.time()
loss_hist = svm.train(X_train, y_train, learning_rate=1e-7, reg=2.5e4
,
                      num_iters=3000, verbose=True)
toc = time.time()
print('That took %fs' % (toc - tic))
```

```
iteration 0 / 3000: loss 781.744439
iteration 100 / 3000: loss 284.328956
iteration 200 / 3000: loss 106.640172
iteration 300 / 3000: loss 41.868259
iteration 400 / 3000: loss 19.077488
iteration 500 / 3000: loss 10.456769
iteration 600 / 3000: loss 6.864472
iteration 700 / 3000: loss 6.292737
iteration 800 / 3000: loss 5.695862
iteration 900 / 3000: loss 5.705327
iteration 1000 / 3000: loss 4.971040
iteration 1100 / 3000: loss 5.027295
iteration 1200 / 3000: loss 5.336624
iteration 1300 / 3000: loss 5.030771
iteration 1400 / 3000: loss 5.618442
iteration 1500 / 3000: loss 5.526525
iteration 1600 / 3000: loss 5.549422
iteration 1700 / 3000: loss 5.443943
iteration 1800 / 3000: loss 5.853622
iteration 1900 / 3000: loss 5.060834
iteration 2000 / 3000: loss 5.019086
iteration 2100 / 3000: loss 4.903166
iteration 2200 / 3000: loss 5.165892
iteration 2300 / 3000: loss 5.042505
iteration 2400 / 3000: loss 6.020268
iteration 2500 / 3000: loss 4.640616
iteration 2600 / 3000: loss 4.889599
iteration 2700 / 3000: loss 5.374185
iteration 2800 / 3000: loss 5.169209
iteration 2900 / 3000: loss 5.492799
That took 4.533447s
```

In [37]:
```
# A useful debugging strategy is to plot the loss as a function of
# iteration number:
plt.plot(loss_hist)
plt.xlabel('Iteration number')
plt.ylabel('Loss value')
plt.show()
```



In [38]:
```
# Write the LinearSVM.predict function and evaluate the performance o
n both the
# 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), ))
```

```
training accuracy: 0.371327
validation accuracy: 0.384000
```

In [39]:
```
# Use the validation set to tune hyperparameters (regularization stre
ngth and
# learning rate). You should experiment with different ranges for the
learning
# rates and regularization strengths; if you are careful you should b
e able to
# get a classification accuracy of about 0.4 on the validation set.
learning_rates = [1e-8, 5e-6]
regularization_strengths = [2e3, 5e4]

# results is dictionary mapping tuples of the form
# (learning_rate, regularization_strength) to tuples of the form
# (training_accuracy, validation_accuracy). The accuracy is simply th
e fraction
# of data points that are correctly classified.
results = {}
best_val = -1   # The highest validation accuracy that we have seen s
o far.
best_svm = None # The LinearSVM object that achieved the highest vali
dation rate.

######################################################################
###########
# TODO:
#
# Write code that chooses the best hyperparameters by tuning on the v
alidation #
# set. For each combination of hyperparameters, train a linear SVM on
the      #
# training set, compute its accuracy on the training and validation s
ets, and  #
# store these numbers in the results dictionary. In addition, store t
he best   #
# validation accuracy in best_val and the LinearSVM object that achie
ves this  #
# accuracy in best_svm.
#
#
#
# Hint: You should use a small value for num_iters as you develop you
r         #
# validation code so that the SVMs don't take much time to train; onc
e you are #
# confident that your validation code works, you should rerun the val
idation   #
# code with a larger value for num_iters.
#
######################################################################
###########
# Your code
# learning_rate = 2e-8
# reg = 4e3
num_iters = 10
for it in range(num_iters):
    for jt in range(num_iters):
        svm = LinearSVM()
```

```
        learning_rate = learning_rates[0] + it * ((learning_rates[1]
- learning_rates[0]) / num_iters)
        reg = regularization_strengths[0] + jt * ((regularization_str
engths[1] - regularization_strengths[0])/ num_iters)
        loss_hist = svm.train(X_train, y_train, learning_rate=learnin
g_rate, reg=reg,
                          num_iters=3000, verbose=False)
        y_train_pred = svm.predict(X_train)
        y_val_pred = svm.predict(X_val)
        training_accuracy = np.mean(y_train == y_train_pred)
        validation_accuracy = np.mean(y_val == y_val_pred)
        results[(learning_rate, reg)] = (training_accuracy, validatio
n_accuracy)
#         reg = reg + 0.05e3
        if validation_accuracy > best_val:
            best_val = validation_accuracy
            best_svm = svm

#####################################################################
###########
#                              END OF YOUR CODE
#
#####################################################################
###########

# Print out results.
for lr, reg in sorted(results):
    train_accuracy, val_accuracy = results[(lr, reg)]
    print('lr %e reg %e train accuracy: %f val accuracy: %f' % (
                lr, reg, train_accuracy, val_accuracy))

print('best validation accuracy achieved during cross-validation: %f'
% best_val)
```

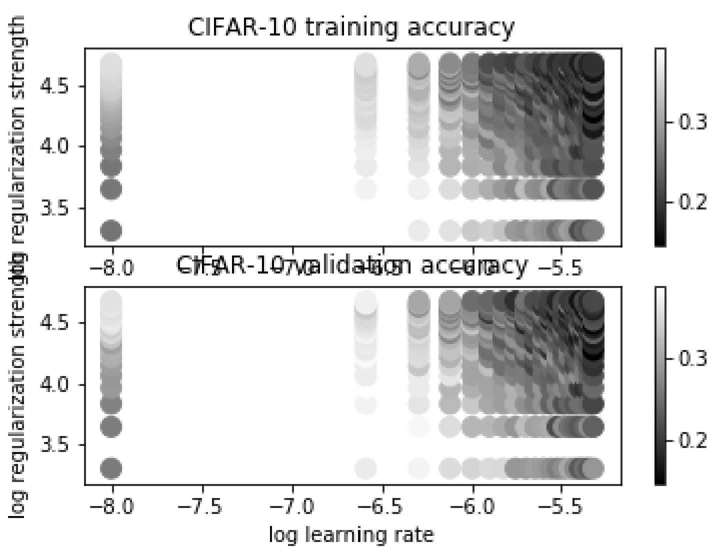
```
lr 1.000000e-08 reg 2.000000e+03 train accuracy: 0.252061 val accurac
y: 0.261000
lr 1.000000e-08 reg 6.800000e+03 train accuracy: 0.260714 val accurac
y: 0.253000
lr 1.000000e-08 reg 1.160000e+04 train accuracy: 0.283347 val accurac
y: 0.291000
lr 1.000000e-08 reg 1.640000e+04 train accuracy: 0.295592 val accurac
y: 0.294000
lr 1.000000e-08 reg 2.120000e+04 train accuracy: 0.319633 val accurac
y: 0.333000
lr 1.000000e-08 reg 2.600000e+04 train accuracy: 0.327694 val accurac
y: 0.336000
lr 1.000000e-08 reg 3.080000e+04 train accuracy: 0.341837 val accurac
y: 0.365000
lr 1.000000e-08 reg 3.560000e+04 train accuracy: 0.350143 val accurac
y: 0.362000
lr 1.000000e-08 reg 4.040000e+04 train accuracy: 0.356755 val accurac
y: 0.368000
lr 1.000000e-08 reg 4.520000e+04 train accuracy: 0.358449 val accurac
y: 0.374000
lr 5.090000e-07 reg 2.000000e+03 train accuracy: 0.378980 val accurac
y: 0.385000
lr 5.090000e-07 reg 6.800000e+03 train accuracy: 0.369755 val accurac
y: 0.374000
lr 5.090000e-07 reg 1.160000e+04 train accuracy: 0.333367 val accurac
y: 0.328000
lr 5.090000e-07 reg 1.640000e+04 train accuracy: 0.338143 val accurac
y: 0.344000
lr 5.090000e-07 reg 2.120000e+04 train accuracy: 0.343633 val accurac
y: 0.357000
lr 5.090000e-07 reg 2.600000e+04 train accuracy: 0.336755 val accurac
y: 0.343000
lr 5.090000e-07 reg 3.080000e+04 train accuracy: 0.335857 val accurac
y: 0.334000
lr 5.090000e-07 reg 3.560000e+04 train accuracy: 0.330980 val accurac
y: 0.339000
lr 5.090000e-07 reg 4.040000e+04 train accuracy: 0.320184 val accurac
y: 0.325000
lr 5.090000e-07 reg 4.520000e+04 train accuracy: 0.326184 val accurac
y: 0.349000
lr 1.008000e-06 reg 2.000000e+03 train accuracy: 0.369388 val accurac
y: 0.391000
lr 1.008000e-06 reg 6.800000e+03 train accuracy: 0.353061 val accurac
y: 0.357000
lr 1.008000e-06 reg 1.160000e+04 train accuracy: 0.312163 val accurac
y: 0.330000
lr 1.008000e-06 reg 1.640000e+04 train accuracy: 0.304776 val accurac
y: 0.299000
lr 1.008000e-06 reg 2.120000e+04 train accuracy: 0.277265 val accurac
y: 0.284000
lr 1.008000e-06 reg 2.600000e+04 train accuracy: 0.252163 val accurac
y: 0.259000
lr 1.008000e-06 reg 3.080000e+04 train accuracy: 0.265633 val accurac
y: 0.283000
lr 1.008000e-06 reg 3.560000e+04 train accuracy: 0.285837 val accurac
y: 0.287000
lr 1.008000e-06 reg 4.040000e+04 train accuracy: 0.292122 val accurac
```

```
y: 0.313000
lr 1.008000e-06 reg 4.520000e+04 train accuracy: 0.261306 val accurac
y: 0.260000
lr 1.507000e-06 reg 2.000000e+03 train accuracy: 0.308102 val accurac
y: 0.300000
lr 1.507000e-06 reg 6.800000e+03 train accuracy: 0.305184 val accurac
y: 0.310000
lr 1.507000e-06 reg 1.160000e+04 train accuracy: 0.252102 val accurac
y: 0.257000
lr 1.507000e-06 reg 1.640000e+04 train accuracy: 0.267469 val accurac
y: 0.297000
lr 1.507000e-06 reg 2.120000e+04 train accuracy: 0.240633 val accurac
y: 0.236000
lr 1.507000e-06 reg 2.600000e+04 train accuracy: 0.261510 val accurac
y: 0.270000
lr 1.507000e-06 reg 3.080000e+04 train accuracy: 0.183633 val accurac
y: 0.186000
lr 1.507000e-06 reg 3.560000e+04 train accuracy: 0.273449 val accurac
y: 0.267000
lr 1.507000e-06 reg 4.040000e+04 train accuracy: 0.232918 val accurac
y: 0.223000
lr 1.507000e-06 reg 4.520000e+04 train accuracy: 0.229755 val accurac
y: 0.234000
lr 2.006000e-06 reg 2.000000e+03 train accuracy: 0.303245 val accurac
y: 0.302000
lr 2.006000e-06 reg 6.800000e+03 train accuracy: 0.293755 val accurac
y: 0.302000
lr 2.006000e-06 reg 1.160000e+04 train accuracy: 0.255776 val accurac
y: 0.238000
lr 2.006000e-06 reg 1.640000e+04 train accuracy: 0.263959 val accurac
y: 0.269000
lr 2.006000e-06 reg 2.120000e+04 train accuracy: 0.244286 val accurac
y: 0.255000
lr 2.006000e-06 reg 2.600000e+04 train accuracy: 0.245204 val accurac
y: 0.253000
lr 2.006000e-06 reg 3.080000e+04 train accuracy: 0.263735 val accurac
y: 0.263000
lr 2.006000e-06 reg 3.560000e+04 train accuracy: 0.222755 val accurac
y: 0.251000
lr 2.006000e-06 reg 4.040000e+04 train accuracy: 0.242347 val accurac
y: 0.259000
lr 2.006000e-06 reg 4.520000e+04 train accuracy: 0.247551 val accurac
y: 0.248000
lr 2.505000e-06 reg 2.000000e+03 train accuracy: 0.302408 val accurac
y: 0.304000
lr 2.505000e-06 reg 6.800000e+03 train accuracy: 0.235204 val accurac
y: 0.239000
lr 2.505000e-06 reg 1.160000e+04 train accuracy: 0.293918 val accurac
y: 0.310000
lr 2.505000e-06 reg 1.640000e+04 train accuracy: 0.222327 val accurac
y: 0.208000
lr 2.505000e-06 reg 2.120000e+04 train accuracy: 0.201347 val accurac
y: 0.209000
lr 2.505000e-06 reg 2.600000e+04 train accuracy: 0.269878 val accurac
y: 0.270000
lr 2.505000e-06 reg 3.080000e+04 train accuracy: 0.232673 val accurac
y: 0.234000
```

```
lr 2.505000e-06 reg 3.560000e+04 train accuracy: 0.193245 val accurac
y: 0.186000
lr 2.505000e-06 reg 4.040000e+04 train accuracy: 0.239265 val accurac
y: 0.249000
lr 2.505000e-06 reg 4.520000e+04 train accuracy: 0.174816 val accurac
y: 0.171000
lr 3.004000e-06 reg 2.000000e+03 train accuracy: 0.288163 val accurac
y: 0.286000
lr 3.004000e-06 reg 6.800000e+03 train accuracy: 0.246122 val accurac
y: 0.257000
lr 3.004000e-06 reg 1.160000e+04 train accuracy: 0.224510 val accurac
y: 0.237000
lr 3.004000e-06 reg 1.640000e+04 train accuracy: 0.230449 val accurac
y: 0.238000
lr 3.004000e-06 reg 2.120000e+04 train accuracy: 0.251122 val accurac
y: 0.241000
lr 3.004000e-06 reg 2.600000e+04 train accuracy: 0.253878 val accurac
y: 0.246000
lr 3.004000e-06 reg 3.080000e+04 train accuracy: 0.195265 val accurac
y: 0.221000
lr 3.004000e-06 reg 3.560000e+04 train accuracy: 0.193633 val accurac
y: 0.189000
lr 3.004000e-06 reg 4.040000e+04 train accuracy: 0.210163 val accurac
y: 0.218000
lr 3.004000e-06 reg 4.520000e+04 train accuracy: 0.199816 val accurac
y: 0.201000
lr 3.503000e-06 reg 2.000000e+03 train accuracy: 0.240204 val accurac
y: 0.248000
lr 3.503000e-06 reg 6.800000e+03 train accuracy: 0.218694 val accurac
y: 0.217000
lr 3.503000e-06 reg 1.160000e+04 train accuracy: 0.242878 val accurac
y: 0.257000
lr 3.503000e-06 reg 1.640000e+04 train accuracy: 0.202816 val accurac
y: 0.221000
lr 3.503000e-06 reg 2.120000e+04 train accuracy: 0.218429 val accurac
y: 0.228000
lr 3.503000e-06 reg 2.600000e+04 train accuracy: 0.205531 val accurac
y: 0.206000
lr 3.503000e-06 reg 3.080000e+04 train accuracy: 0.213694 val accurac
y: 0.223000
lr 3.503000e-06 reg 3.560000e+04 train accuracy: 0.211980 val accurac
y: 0.213000
lr 3.503000e-06 reg 4.040000e+04 train accuracy: 0.184449 val accurac
y: 0.192000
lr 3.503000e-06 reg 4.520000e+04 train accuracy: 0.210510 val accurac
y: 0.200000
lr 4.002000e-06 reg 2.000000e+03 train accuracy: 0.255531 val accurac
y: 0.280000
lr 4.002000e-06 reg 6.800000e+03 train accuracy: 0.231959 val accurac
y: 0.227000
lr 4.002000e-06 reg 1.160000e+04 train accuracy: 0.206878 val accurac
y: 0.193000
lr 4.002000e-06 reg 1.640000e+04 train accuracy: 0.175184 val accurac
y: 0.190000
lr 4.002000e-06 reg 2.120000e+04 train accuracy: 0.210082 val accurac
y: 0.211000
lr 4.002000e-06 reg 2.600000e+04 train accuracy: 0.164531 val accurac
```

```
y: 0.157000
lr 4.002000e-06 reg 3.080000e+04 train accuracy: 0.184265 val accurac
y: 0.172000
lr 4.002000e-06 reg 3.560000e+04 train accuracy: 0.196918 val accurac
y: 0.198000
lr 4.002000e-06 reg 4.040000e+04 train accuracy: 0.178347 val accurac
y: 0.188000
lr 4.002000e-06 reg 4.520000e+04 train accuracy: 0.188816 val accurac
y: 0.189000
lr 4.501000e-06 reg 2.000000e+03 train accuracy: 0.255449 val accurac
y: 0.243000
lr 4.501000e-06 reg 6.800000e+03 train accuracy: 0.279224 val accurac
y: 0.283000
lr 4.501000e-06 reg 1.160000e+04 train accuracy: 0.223102 val accurac
y: 0.260000
lr 4.501000e-06 reg 1.640000e+04 train accuracy: 0.179531 val accurac
y: 0.187000
lr 4.501000e-06 reg 2.120000e+04 train accuracy: 0.153020 val accurac
y: 0.157000
lr 4.501000e-06 reg 2.600000e+04 train accuracy: 0.191367 val accurac
y: 0.184000
lr 4.501000e-06 reg 3.080000e+04 train accuracy: 0.172020 val accurac
y: 0.182000
lr 4.501000e-06 reg 3.560000e+04 train accuracy: 0.209143 val accurac
y: 0.196000
lr 4.501000e-06 reg 4.040000e+04 train accuracy: 0.191327 val accurac
y: 0.193000
lr 4.501000e-06 reg 4.520000e+04 train accuracy: 0.157776 val accurac
y: 0.168000
best validation accuracy achieved during cross-validation: 0.391000
```

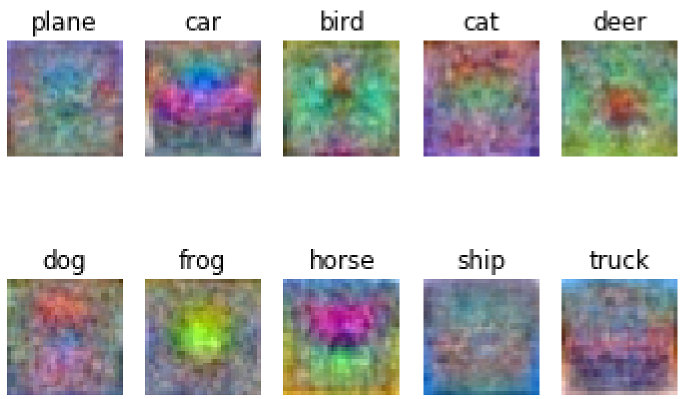
In [20]:
```
# Visualize the cross-validation results
import math
x_scatter = [math.log10(x[0]) for x in results]
y_scatter = [math.log10(x[1]) for x in results]

# plot training accuracy
marker_size = 100
colors = [results[x][0] for x in results]
plt.subplot(2, 1, 1)
plt.scatter(x_scatter, y_scatter, marker_size, c=colors)
plt.colorbar()
plt.xlabel('log learning rate')
plt.ylabel('log regularization strength')
plt.title('CIFAR-10 training accuracy')

# plot validation accuracy
colors = [results[x][1] for x in results] # default size of markers i
s 20
plt.subplot(2, 1, 2)
plt.scatter(x_scatter, y_scatter, marker_size, c=colors)
plt.colorbar()
plt.xlabel('log learning rate')
plt.ylabel('log regularization strength')
plt.title('CIFAR-10 validation accuracy')
plt.show()
```



In [21]:
```
# 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_a
ccuracy)
```

```
linear SVM on raw pixels final test set accuracy: 0.368000
```

In [22]:
```
# Visualize the learned weights for each class.
# Depending on your choice of learning rate and regularization streng
th, these may
# or may not be nice to look at.
w = best_svm.W[:-1,:] # strip out the bias
w = w.reshape(32, 32, 3, 10)
w_min, w_max = np.min(w), np.max(w)
classes = ['plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'hor
se', 'ship', 'truck']
for i in range(10):
    plt.subplot(2, 5, i + 1)

    # Rescale the weights to be between 0 and 255
    wimg = 255.0 * (w[:, :, :, i].squeeze() - w_min) / (w_max - w_min
)
    plt.imshow(wimg.astype('uint8'))
    plt.axis('off')
    plt.title(classes[i])
```



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: *The visualized SVM weights look like the average of all the images in that category. This is because we are effectively doing a cos function of two vectors, one being the visualized SVM and the other being test. To get best possible result for cos, they need to be as close to each other as possible, which means the visualized SVM should give maximum result when we do a cos function with that category images. An average of the images in this category fits this category reasonably well.*

Softmax exercise

Complete and hand in this completed worksheet (including its outputs and any supporting code outside of the worksheet) with your assignment submission. For more details see the [assignments page (https://compsci682-fa19.github.io/assignments2019/assignment1/)](https://compsci682-fa19.github.io/assignments2019/assignment1/) on the course website.

This exercise is analogous to the SVM exercise. You will:

- implement a fully-vectorized **loss function** for the Softmax classifier
- implement the fully-vectorized expression for its **analytic gradient**
- **check your implementation** with numerical gradient
- use a validation set to **tune the learning rate and regularization** strength
- **optimize** the loss function with **SGD**
- **visualize** the final learned weights

In [1]:
```
from __future__ import print_function
import random
import numpy as np
from cs682.data_utils import load_CIFAR10
import matplotlib.pyplot as plt


%matplotlib inline
plt.rcParams['figure.figsize'] = (10.0, 8.0) # set default size of pl
ots
plt.rcParams['image.interpolation'] = 'nearest'
plt.rcParams['image.cmap'] = 'gray'

# for auto-reloading extenrnal modules
# see http://stackoverflow.com/questions/1907993/autoreload-of-module
s-in-ipython
%load_ext autoreload
%autoreload 2
```

In [2]:
```
def get_CIFAR10_data(num_training=49000, num_validation=1000, num_tes
t=1000, num_dev=500):
    """
    Load the CIFAR-10 dataset from disk and perform preprocessing to
 prepare
    it for the linear classifier. These are the same steps as we used
for the
    SVM, but condensed to a single function.
    """
    # Load the raw CIFAR-10 data
    cifar10_dir = 'cs682/datasets/cifar-10-batches-py'

    X_train, y_train, X_test, y_test = load_CIFAR10(cifar10_dir)

    # subsample the data
    mask = list(range(num_training, num_training + num_validation))
    X_val = X_train[mask]
    y_val = y_train[mask]
    mask = list(range(num_training))
    X_train = X_train[mask]
    y_train = y_train[mask]
    mask = list(range(num_test))
    X_test = X_test[mask]
    y_test = y_test[mask]
    mask = np.random.choice(num_training, num_dev, replace=False)
    X_dev = X_train[mask]
    y_dev = y_train[mask]

    # Preprocessing: reshape the image data into rows
    X_train = np.reshape(X_train, (X_train.shape[0], -1))
    X_val = np.reshape(X_val, (X_val.shape[0], -1))
    X_test = np.reshape(X_test, (X_test.shape[0], -1))
    X_dev = np.reshape(X_dev, (X_dev.shape[0], -1))

    # Normalize the data: subtract the mean image
    mean_image = np.mean(X_train, axis = 0)
    X_train -= mean_image
    X_val -= mean_image
    X_test -= mean_image
    X_dev -= mean_image

    # add bias dimension and transform into columns
    X_train = np.hstack([X_train, np.ones((X_train.shape[0], 1))])
    X_val = np.hstack([X_val, np.ones((X_val.shape[0], 1))])
    X_test = np.hstack([X_test, np.ones((X_test.shape[0], 1))])
    X_dev = np.hstack([X_dev, np.ones((X_dev.shape[0], 1))])

    return X_train, y_train, X_val, y_val, X_test, y_test, X_dev, y_d
ev


# Cleaning up variables to prevent loading data multiple times (which
may cause memory issue)
try:
   del X_train, y_train
   del X_test, y_test
```

```
    print('Clear previously loaded data.')
except:
    pass

# Invoke the above function to get our data.
X_train, y_train, X_val, y_val, X_test, y_test, X_dev, y_dev = get_CI
FAR10_data()
print('Train data shape: ', X_train.shape)
print('Train labels shape: ', y_train.shape)
print('Validation data shape: ', X_val.shape)
print('Validation labels shape: ', y_val.shape)
print('Test data shape: ', X_test.shape)
print('Test labels shape: ', y_test.shape)
print('dev data shape: ', X_dev.shape)
print('dev labels shape: ', y_dev.shape)
```

```
Train data shape:  (49000, 3073)
Train labels shape:  (49000,)
Validation data shape:  (1000, 3073)
Validation labels shape:  (1000,)
Test data shape:  (1000, 3073)
Test labels shape:  (1000,)
dev data shape:  (500, 3073)
dev labels shape:  (500,)
```

Softmax Classifier

Your code for this section will all be written inside **cs682/classifiers/softmax.py**.

In [3]:
```
# First implement the naive softmax loss function with nested loops.
# Open the file cs682/classifiers/softmax.py and implement the
# softmax_loss_naive function.

from cs682.classifiers.softmax import softmax_loss_naive
import time

# Generate a random softmax weight matrix and use it to compute the l
oss.
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
(0.1).
print('loss: %f' % loss)
print('sanity check: %f' % (-np.log(0.1)))
```

```
loss: 2.340098
sanity check: 2.302585
```

Inline Question 1:

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

Your answer: We are initiating elements of W to be very small. Hence, when we take exponent of a very small number, we get exp(~0) => exp(0) = 1. In softmax function, if we approximate all the terms to 1, then we end up with 1/10 for each exponent of score. Essentially, we get -log(1/10) = -log(0.1). Since it's the same for every example, even if we average it over, we get -log(0.1).

In [4]:
```
# Complete the implementation of softmax_loss_naive and implement a
 (naive)
# 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
tool.
# The numeric gradient should be close to the analytic gradient.
from cs682.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)
```

```
numerical: 3.902427 analytic: 3.902427, relative error: 1.351823e-08
numerical: -2.169901 analytic: -2.169901, relative error: 9.049346e-0
9
numerical: 1.845966 analytic: 1.845966, relative error: 3.389752e-08
numerical: 2.336157 analytic: 2.336157, relative error: 1.254159e-08
numerical: -4.579238 analytic: -4.579238, relative error: 3.704144e-0
9
numerical: 1.454987 analytic: 1.454987, relative error: 9.621803e-09
numerical: -0.377293 analytic: -0.377293, relative error: 1.466624e-0
7
numerical: -0.012592 analytic: -0.012592, relative error: 1.709682e-0
6
numerical: -0.005379 analytic: -0.005379, relative error: 7.520723e-0
6
numerical: 2.694451 analytic: 2.694451, relative error: 9.489678e-09
numerical: -0.440033 analytic: -0.440033, relative error: 1.006129e-0
8
numerical: 0.960237 analytic: 0.960237, relative error: 4.966151e-08
numerical: 0.846898 analytic: 0.846898, relative error: 7.480986e-08
numerical: 2.486037 analytic: 2.486037, relative error: 2.742079e-08
numerical: 0.244251 analytic: 0.244251, relative error: 1.832118e-07
numerical: 1.300372 analytic: 1.300372, relative error: 3.288925e-08
numerical: -0.349705 analytic: -0.349705, relative error: 7.449021e-0
8
numerical: -0.803577 analytic: -0.803577, relative error: 3.878746e-0
8
numerical: 0.636018 analytic: 0.636018, relative error: 3.871182e-08
numerical: -0.159012 analytic: -0.159012, relative error: 1.621705e-0
7
```

In [5]:
```
# Now that we have a naive implementation of the softmax loss functio
n and its gradient,
# implement a vectorized version in softmax_loss_vectorized.
# The two versions should compute the same results, but the vectorize
d version should be
# much faster.
tic = time.time()
loss_naive, grad_naive = softmax_loss_naive(W, X_dev, y_dev, 0.000005
)
toc = time.time()
print('naive loss: %e computed in %fs' % (loss_naive, toc - tic))

from cs682.classifiers.softmax import softmax_loss_vectorized
tic = time.time()
loss_vectorized, grad_vectorized = softmax_loss_vectorized(W, X_dev,
y_dev, 0.000005)
toc = time.time()
print('vectorized loss: %e computed in %fs' % (loss_vectorized, toc -
tic))

# As we did for the SVM, we use the Frobenius norm to compare the two
versions
# of the gradient.
grad_difference = np.linalg.norm(grad_naive - grad_vectorized, ord='f
ro')
print('Loss difference: %f' % np.abs(loss_naive - loss_vectorized))
print('Gradient difference: %f' % grad_difference)
```

```
naive loss: 2.340098e+00 computed in 0.084918s
vectorized loss: 2.340098e+00 computed in 0.002776s
Loss difference: 0.000000
Gradient difference: 0.000000
```

In [7]:
```
# Use the validation set to tune hyperparameters (regularization stre
ngth and
# learning rate). You should experiment with different ranges for the
learning
# rates and regularization strengths; if you are careful you should b
e able to
# get a classification accuracy of over 0.35 on the validation set.
from cs682.classifiers import Softmax
results = {}
best_val = -1
best_softmax = None
learning_rates = [1e-8, 5e-7]
regularization_strengths = [2.5e4, 5e4]

####################################################################
###########
# TODO:
#
# Use the validation set to set the learning rate and regularization
 strength. #
# This should be identical to the validation that you did for the SV
M; save    #
# the best trained softmax classifer in best_softmax.
#
####################################################################
###########
num_iters = 10
for it in range(num_iters):
    for jt in range(num_iters):
        softmax = Softmax()
        learning_rate = learning_rates[0] + it * ((learning_rates[1]
- learning_rates[0]) / num_iters)
        reg = regularization_strengths[0] + jt * ((regularization_str
engths[1] - regularization_strengths[0])/ num_iters)
        loss_hist = softmax.train(X_train, y_train, learning_rate=lea
rning_rate, reg=reg,
                              num_iters=3000, verbose=False)

        y_train_pred = softmax.predict(X_train)
        y_val_pred = softmax.predict(X_val)
        training_accuracy = np.mean(y_train == y_train_pred)
        validation_accuracy = np.mean(y_val == y_val_pred)
        results[(learning_rate, reg)] = (training_accuracy, validatio
n_accuracy)
        if validation_accuracy > best_val:
            best_val = validation_accuracy
            best_softmax = softmax

####################################################################
###########
#                              END OF YOUR CODE
#
####################################################################
###########

# Print out results.
```

```
for lr, reg in sorted(results):
    train_accuracy, val_accuracy = results[(lr, reg)]
    print('lr %e reg %e train accuracy: %f val accuracy: %f' % (
                lr, reg, train_accuracy, val_accuracy))

print('best validation accuracy achieved during cross-validation: %f' % best_val)
```

```
lr 1.000000e-08 reg 2.500000e+04 train accuracy: 0.263367 val accurac
y: 0.275000
lr 1.000000e-08 reg 2.750000e+04 train accuracy: 0.261469 val accurac
y: 0.257000
lr 1.000000e-08 reg 3.000000e+04 train accuracy: 0.258673 val accurac
y: 0.270000
lr 1.000000e-08 reg 3.250000e+04 train accuracy: 0.262061 val accurac
y: 0.269000
lr 1.000000e-08 reg 3.500000e+04 train accuracy: 0.274041 val accurac
y: 0.305000
lr 1.000000e-08 reg 3.750000e+04 train accuracy: 0.288204 val accurac
y: 0.306000
lr 1.000000e-08 reg 4.000000e+04 train accuracy: 0.283612 val accurac
y: 0.297000
lr 1.000000e-08 reg 4.250000e+04 train accuracy: 0.295265 val accurac
y: 0.311000
lr 1.000000e-08 reg 4.500000e+04 train accuracy: 0.293816 val accurac
y: 0.301000
lr 1.000000e-08 reg 4.750000e+04 train accuracy: 0.291673 val accurac
y: 0.281000
lr 5.900000e-08 reg 2.500000e+04 train accuracy: 0.325327 val accurac
y: 0.338000
lr 5.900000e-08 reg 2.750000e+04 train accuracy: 0.326388 val accurac
y: 0.341000
lr 5.900000e-08 reg 3.000000e+04 train accuracy: 0.326429 val accurac
y: 0.340000
lr 5.900000e-08 reg 3.250000e+04 train accuracy: 0.323653 val accurac
y: 0.337000
lr 5.900000e-08 reg 3.500000e+04 train accuracy: 0.319531 val accurac
y: 0.334000
lr 5.900000e-08 reg 3.750000e+04 train accuracy: 0.321755 val accurac
y: 0.336000
lr 5.900000e-08 reg 4.000000e+04 train accuracy: 0.307571 val accurac
y: 0.328000
lr 5.900000e-08 reg 4.250000e+04 train accuracy: 0.307551 val accurac
y: 0.322000
lr 5.900000e-08 reg 4.500000e+04 train accuracy: 0.310224 val accurac
y: 0.327000
lr 5.900000e-08 reg 4.750000e+04 train accuracy: 0.304286 val accurac
y: 0.323000
lr 1.080000e-07 reg 2.500000e+04 train accuracy: 0.330469 val accurac
y: 0.350000
lr 1.080000e-07 reg 2.750000e+04 train accuracy: 0.325612 val accurac
y: 0.336000
lr 1.080000e-07 reg 3.000000e+04 train accuracy: 0.319612 val accurac
y: 0.335000
lr 1.080000e-07 reg 3.250000e+04 train accuracy: 0.314571 val accurac
y: 0.329000
lr 1.080000e-07 reg 3.500000e+04 train accuracy: 0.322959 val accurac
y: 0.327000
lr 1.080000e-07 reg 3.750000e+04 train accuracy: 0.308837 val accurac
y: 0.319000
lr 1.080000e-07 reg 4.000000e+04 train accuracy: 0.316163 val accurac
y: 0.329000
lr 1.080000e-07 reg 4.250000e+04 train accuracy: 0.315898 val accurac
y: 0.330000
lr 1.080000e-07 reg 4.500000e+04 train accuracy: 0.317490 val accurac
```

```
y: 0.334000
lr 1.080000e-07 reg 4.750000e+04 train accuracy: 0.311612 val accurac
y: 0.325000
lr 1.570000e-07 reg 2.500000e+04 train accuracy: 0.331857 val accurac
y: 0.344000
lr 1.570000e-07 reg 2.750000e+04 train accuracy: 0.329612 val accurac
y: 0.339000
lr 1.570000e-07 reg 3.000000e+04 train accuracy: 0.316510 val accurac
y: 0.336000
lr 1.570000e-07 reg 3.250000e+04 train accuracy: 0.316367 val accurac
y: 0.340000
lr 1.570000e-07 reg 3.500000e+04 train accuracy: 0.320592 val accurac
y: 0.328000
lr 1.570000e-07 reg 3.750000e+04 train accuracy: 0.320959 val accurac
y: 0.333000
lr 1.570000e-07 reg 4.000000e+04 train accuracy: 0.313571 val accurac
y: 0.330000
lr 1.570000e-07 reg 4.250000e+04 train accuracy: 0.312449 val accurac
y: 0.328000
lr 1.570000e-07 reg 4.500000e+04 train accuracy: 0.312102 val accurac
y: 0.324000
lr 1.570000e-07 reg 4.750000e+04 train accuracy: 0.309510 val accurac
y: 0.323000
lr 2.060000e-07 reg 2.500000e+04 train accuracy: 0.330082 val accurac
y: 0.340000
lr 2.060000e-07 reg 2.750000e+04 train accuracy: 0.327980 val accurac
y: 0.350000
lr 2.060000e-07 reg 3.000000e+04 train accuracy: 0.325673 val accurac
y: 0.327000
lr 2.060000e-07 reg 3.250000e+04 train accuracy: 0.311286 val accurac
y: 0.327000
lr 2.060000e-07 reg 3.500000e+04 train accuracy: 0.316714 val accurac
y: 0.334000
lr 2.060000e-07 reg 3.750000e+04 train accuracy: 0.322102 val accurac
y: 0.336000
lr 2.060000e-07 reg 4.000000e+04 train accuracy: 0.310796 val accurac
y: 0.333000
lr 2.060000e-07 reg 4.250000e+04 train accuracy: 0.310857 val accurac
y: 0.329000
lr 2.060000e-07 reg 4.500000e+04 train accuracy: 0.314449 val accurac
y: 0.331000
lr 2.060000e-07 reg 4.750000e+04 train accuracy: 0.308980 val accurac
y: 0.328000
lr 2.550000e-07 reg 2.500000e+04 train accuracy: 0.333082 val accurac
y: 0.349000
lr 2.550000e-07 reg 2.750000e+04 train accuracy: 0.329592 val accurac
y: 0.339000
lr 2.550000e-07 reg 3.000000e+04 train accuracy: 0.311163 val accurac
y: 0.338000
lr 2.550000e-07 reg 3.250000e+04 train accuracy: 0.326837 val accurac
y: 0.347000
lr 2.550000e-07 reg 3.500000e+04 train accuracy: 0.314898 val accurac
y: 0.337000
lr 2.550000e-07 reg 3.750000e+04 train accuracy: 0.308265 val accurac
y: 0.325000
lr 2.550000e-07 reg 4.000000e+04 train accuracy: 0.308531 val accurac
y: 0.317000
```

```
lr 2.550000e-07 reg 4.250000e+04 train accuracy: 0.314122 val accurac
y: 0.322000
lr 2.550000e-07 reg 4.500000e+04 train accuracy: 0.306184 val accurac
y: 0.327000
lr 2.550000e-07 reg 4.750000e+04 train accuracy: 0.305959 val accurac
y: 0.323000
lr 3.040000e-07 reg 2.500000e+04 train accuracy: 0.335796 val accurac
y: 0.344000
lr 3.040000e-07 reg 2.750000e+04 train accuracy: 0.328959 val accurac
y: 0.336000
lr 3.040000e-07 reg 3.000000e+04 train accuracy: 0.313224 val accurac
y: 0.329000
lr 3.040000e-07 reg 3.250000e+04 train accuracy: 0.306837 val accurac
y: 0.320000
lr 3.040000e-07 reg 3.500000e+04 train accuracy: 0.313612 val accurac
y: 0.325000
lr 3.040000e-07 reg 3.750000e+04 train accuracy: 0.315184 val accurac
y: 0.330000
lr 3.040000e-07 reg 4.000000e+04 train accuracy: 0.299490 val accurac
y: 0.322000
lr 3.040000e-07 reg 4.250000e+04 train accuracy: 0.307796 val accurac
y: 0.338000
lr 3.040000e-07 reg 4.500000e+04 train accuracy: 0.305531 val accurac
y: 0.326000
lr 3.040000e-07 reg 4.750000e+04 train accuracy: 0.292571 val accurac
y: 0.314000
lr 3.530000e-07 reg 2.500000e+04 train accuracy: 0.329408 val accurac
y: 0.337000
lr 3.530000e-07 reg 2.750000e+04 train accuracy: 0.325857 val accurac
y: 0.346000
lr 3.530000e-07 reg 3.000000e+04 train accuracy: 0.317551 val accurac
y: 0.331000
lr 3.530000e-07 reg 3.250000e+04 train accuracy: 0.314531 val accurac
y: 0.345000
lr 3.530000e-07 reg 3.500000e+04 train accuracy: 0.322980 val accurac
y: 0.336000
lr 3.530000e-07 reg 3.750000e+04 train accuracy: 0.321857 val accurac
y: 0.332000
lr 3.530000e-07 reg 4.000000e+04 train accuracy: 0.319143 val accurac
y: 0.327000
lr 3.530000e-07 reg 4.250000e+04 train accuracy: 0.319837 val accurac
y: 0.340000
lr 3.530000e-07 reg 4.500000e+04 train accuracy: 0.307857 val accurac
y: 0.319000
lr 3.530000e-07 reg 4.750000e+04 train accuracy: 0.303306 val accurac
y: 0.312000
lr 4.020000e-07 reg 2.500000e+04 train accuracy: 0.329490 val accurac
y: 0.339000
lr 4.020000e-07 reg 2.750000e+04 train accuracy: 0.329184 val accurac
y: 0.347000
lr 4.020000e-07 reg 3.000000e+04 train accuracy: 0.311041 val accurac
y: 0.325000
lr 4.020000e-07 reg 3.250000e+04 train accuracy: 0.311306 val accurac
y: 0.325000
lr 4.020000e-07 reg 3.500000e+04 train accuracy: 0.316041 val accurac
y: 0.329000
lr 4.020000e-07 reg 3.750000e+04 train accuracy: 0.323592 val accurac
```

```
y: 0.335000
lr 4.020000e-07 reg 4.000000e+04 train accuracy: 0.312347 val accurac
y: 0.327000
lr 4.020000e-07 reg 4.250000e+04 train accuracy: 0.305918 val accurac
y: 0.318000
lr 4.020000e-07 reg 4.500000e+04 train accuracy: 0.308531 val accurac
y: 0.323000
lr 4.020000e-07 reg 4.750000e+04 train accuracy: 0.309878 val accurac
y: 0.319000
lr 4.510000e-07 reg 2.500000e+04 train accuracy: 0.322449 val accurac
y: 0.333000
lr 4.510000e-07 reg 2.750000e+04 train accuracy: 0.329714 val accurac
y: 0.347000
lr 4.510000e-07 reg 3.000000e+04 train accuracy: 0.321061 val accurac
y: 0.332000
lr 4.510000e-07 reg 3.250000e+04 train accuracy: 0.305776 val accurac
y: 0.321000
lr 4.510000e-07 reg 3.500000e+04 train accuracy: 0.310490 val accurac
y: 0.323000
lr 4.510000e-07 reg 3.750000e+04 train accuracy: 0.316796 val accurac
y: 0.328000
lr 4.510000e-07 reg 4.000000e+04 train accuracy: 0.312612 val accurac
y: 0.324000
lr 4.510000e-07 reg 4.250000e+04 train accuracy: 0.294061 val accurac
y: 0.306000
lr 4.510000e-07 reg 4.500000e+04 train accuracy: 0.301694 val accurac
y: 0.320000
lr 4.510000e-07 reg 4.750000e+04 train accuracy: 0.307633 val accurac
y: 0.312000
best validation accuracy achieved during cross-validation: 0.350000
```

In [8]:

```
# 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_acc
uracy, ))
```

```
softmax on raw pixels final test set accuracy: 0.339000
```

Inline Question - *True or False*

It's 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: We have accounted for numerical instability for Softmax because the scores of some points might be very large and when we do an exp on that number, it tends to go to infinity. In case of SVM, the loss remains unchanged because its a max function of difference of scores. But in case of Softmax, if the score is too large, loss changes by a lot because there is exponential function in the loss.

In [9]:
```
# Visualize the learned weights for each class
w = best_softmax.W[:-1,:] # strip out the bias
w = w.reshape(32, 32, 3, 10)

w_min, w_max = np.min(w), np.max(w)

classes = ['plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']
for i in range(10):
    plt.subplot(2, 5, i + 1)

    # Rescale the weights to be between 0 and 255
    wimg = 255.0 * (w[:, :, :, i].squeeze() - w_min) / (w_max - w_min)
    plt.imshow(wimg.astype('uint8'))
    plt.axis('off')
    plt.title(classes[i])
```

In [ ]:
```
```

Implementing a Neural Network

In this exercise we will develop a neural network with fully-connected layers to perform classification, and test it out on the CIFAR-10 dataset.

In [20]:
```
# A bit of setup

from __future__ import print_function
import numpy as np
import matplotlib.pyplot as plt

from cs682.classifiers.neural_net import TwoLayerNet


%matplotlib inline
plt.rcParams['figure.figsize'] = (10.0, 8.0) # set default size of pl
ots
plt.rcParams['image.interpolation'] = 'nearest'
plt.rcParams['image.cmap'] = 'gray'

# for auto-reloading external modules
# see http://stackoverflow.com/questions/1907993/autoreload-of-module
s-in-ipython
%load_ext autoreload
%autoreload 2

def rel_error(x, y):
    """ returns relative error """
    return np.max(np.abs(x - y) / (np.maximum(1e-8, np.abs(x) + np.ab
s(y))))
```

```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
```

We will use the class `TwoLayerNet` in the file `cs682/classifiers/neural_net.py` to represent instances of our network. The network parameters are stored in the instance variable `self.params` where keys are string parameter names and values are numpy arrays. Below, we initialize toy data and a toy model that we will use to develop your implementation.

In [21]:
```
# Create a small net and some toy data to check your implementations.
# Note that we set the random seed for repeatable experiments.

input_size = 4
hidden_size = 10
num_classes = 3
num_inputs = 5

def init_toy_model():
    np.random.seed(0)
    return TwoLayerNet(input_size, hidden_size, num_classes, std=1e-1
)

def init_toy_data():
    np.random.seed(1)
    X = 10 * np.random.randn(num_inputs, input_size)
    y = np.array([0, 1, 2, 2, 1])
    return X, y

net = init_toy_model()
X, y = init_toy_data()
```

Forward pass: compute scores

Open the file `cs682/classifiers/neural_net.py` and look at the method `TwoLayerNet.loss` . This function is very similar to the loss functions you have written for the SVM and Softmax exercises: It takes the data and weights and computes the class scores, the loss, and the gradients on the parameters.

Implement the first part of the forward pass which uses the weights and biases to compute the scores for all inputs.

In [22]:
```
scores = net.loss(X)
print('Your scores:')
print(scores)
print()
print('correct scores:')
correct_scores = np.asarray([
  [-0.81233741, -1.27654624, -0.70335995],
  [-0.17129677, -1.18803311, -0.47310444],
  [-0.51590475, -1.01354314, -0.8504215 ],
  [-0.15419291, -0.48629638, -0.52901952],
  [-0.00618733, -0.12435261, -0.15226949]])
print(correct_scores)
print()

# The difference should be very small. We get < 1e-7
print('Difference between your scores and correct scores:')
print(np.sum(np.abs(scores - correct_scores)))
```

```
Your scores:
[[-0.81233741 -1.27654624 -0.70335995]
 [-0.17129677 -1.18803311 -0.47310444]
 [-0.51590475 -1.01354314 -0.8504215 ]
 [-0.15419291 -0.48629638 -0.52901952]
 [-0.00618733 -0.12435261 -0.15226949]]

correct scores:
[[-0.81233741 -1.27654624 -0.70335995]
 [-0.17129677 -1.18803311 -0.47310444]
 [-0.51590475 -1.01354314 -0.8504215 ]
 [-0.15419291 -0.48629638 -0.52901952]
 [-0.00618733 -0.12435261 -0.15226949]]

Difference between your scores and correct scores:
3.6802720496109664e-08
```

Forward pass: compute loss

In the same function, implement the second part that computes the data and regularizaion loss.

In [24]:
```
loss, _ = net.loss(X, y, reg=0.05)
correct_loss = 1.30378789133

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

```
Difference between your loss and correct loss:
1.794120407794253e-13
```

Backward pass

Implement the rest of the function. This will compute the gradient of the loss with respect to the variables `W1`, `b1`, `W2`, and `b2`. Now that you (hopefully!) have a correctly implemented forward pass, you can debug your backward pass using a numeric gradient check:

In [25]:
```
from cs682.gradient_check import eval_numerical_gradient

# Use numeric gradient checking to check your implementation of the b
ackward pass.
# If your implementation is correct, the difference between the numer
ic and
# analytic gradients should be less than 1e-8 for each of W1, W2, b1,
and b2.

loss, grads = net.loss(X, y, reg=0.05)

# these should all be less than 1e-8 or so
for param_name in grads:
    f = lambda W: net.loss(X, y, reg=0.05)[0]
    param_grad_num = eval_numerical_gradient(f, net.params[param_name
], verbose=False)
    print('%s max relative error: %e' % (param_name, rel_error(param_
grad_num, grads[param_name])))
```

```
W1 max relative error: 3.561318e-09
W2 max relative error: 3.440708e-09
b1 max relative error: 2.738421e-09
b2 max relative error: 3.865070e-11
```

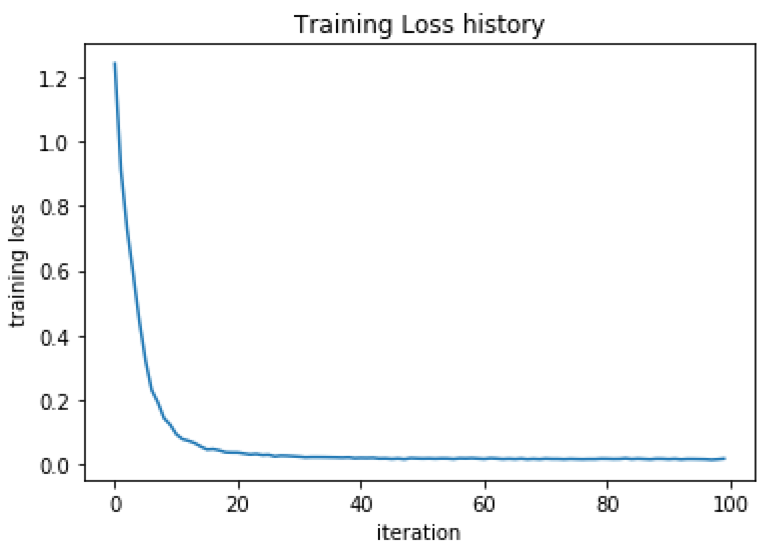
Train the network

To train the network we will use stochastic gradient descent (SGD), similar to the SVM and Softmax classifiers. Look at the function `TwoLayerNet.train` and fill in the missing sections to implement the training procedure. This should be very similar to the training procedure you used for the SVM and Softmax classifiers. You will also have to implement `TwoLayerNet.predict`, as the training process periodically performs prediction to keep track of accuracy over time while the network trains.

Once you have implemented the method, run the code below to train a two-layer network on toy data. You should achieve a training loss less than 0.2.

In [8]:
```
net = init_toy_model()
stats = net.train(X, y, X, y,
            learning_rate=1e-1, reg=5e-6,
            num_iters=100, verbose=False)

print('Final training loss: ', stats['loss_history'][-1])

# plot the loss history
plt.plot(stats['loss_history'])
plt.xlabel('iteration')
plt.ylabel('training loss')
plt.title('Training Loss history')
plt.show()
```

```
Final training loss:  0.017149607938732037
```



Load the data

Now that you have implemented a two-layer network that passes gradient checks and works on toy data, it's time to load up our favorite CIFAR-10 data so we can use it to train a classifier on a real dataset.

In [9]:
```
from cs682.data_utils import load_CIFAR10

def get_CIFAR10_data(num_training=49000, num_validation=1000, num_tes
t=1000):
    """
    Load the CIFAR-10 dataset from disk and perform preprocessing to
 prepare
    it for the two-layer neural net classifier. These are the same st
eps as
    we used for the SVM, but condensed to a single function.
    """
    # Load the raw CIFAR-10 data
    cifar10_dir = 'cs682/datasets/cifar-10-batches-py'

    X_train, y_train, X_test, y_test = load_CIFAR10(cifar10_dir)

    # Subsample the data
    mask = list(range(num_training, num_training + num_validation))
    X_val = X_train[mask]
    y_val = y_train[mask]
    mask = list(range(num_training))
    X_train = X_train[mask]
    y_train = y_train[mask]
    mask = list(range(num_test))
    X_test = X_test[mask]
    y_test = y_test[mask]

    # Normalize the data: subtract the mean image
    mean_image = np.mean(X_train, axis=0)
    X_train -= mean_image
    X_val -= mean_image
    X_test -= mean_image

    # Reshape data to rows
    X_train = X_train.reshape(num_training, -1)
    X_val = X_val.reshape(num_validation, -1)
    X_test = X_test.reshape(num_test, -1)

    return X_train, y_train, X_val, y_val, X_test, y_test


# Cleaning up variables to prevent loading data multiple times (which
may cause memory issue)
try:
   del X_train, y_train
   del X_test, y_test
   print('Clear previously loaded data.')
except:
   pass

# Invoke the above function to get our data.
X_train, y_train, X_val, y_val, X_test, y_test = get_CIFAR10_data()
print('Train data shape: ', X_train.shape)
print('Train labels shape: ', y_train.shape)
print('Validation data shape: ', X_val.shape)
print('Validation labels shape: ', y_val.shape)
```

```
print('Test data shape: ', X_test.shape)
print('Test labels shape: ', y_test.shape)
```

```
Train data shape:  (49000, 3072)
Train labels shape:  (49000,)
Validation data shape:  (1000, 3072)
Validation labels shape:  (1000,)
Test data shape:  (1000, 3072)
Test labels shape:  (1000,)
```

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.

In [26]:
```
input_size = 32 * 32 * 3
hidden_size = 50
num_classes = 10
net = TwoLayerNet(input_size, hidden_size, num_classes)

# Train the network
stats = net.train(X_train, y_train, X_val, y_val,
            num_iters=3000, batch_size=200,
            learning_rate=1e-4, learning_rate_decay=0.95,
            reg=0.25, verbose=True)

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

```
iteration 0 / 3000: loss 2.302973
iteration 100 / 3000: loss 2.302542
iteration 200 / 3000: loss 2.298325
iteration 300 / 3000: loss 2.255561
iteration 400 / 3000: loss 2.182325
iteration 500 / 3000: loss 2.131531
iteration 600 / 3000: loss 2.126443
iteration 700 / 3000: loss 2.063506
iteration 800 / 3000: loss 2.040525
iteration 900 / 3000: loss 1.990341
iteration 1000 / 3000: loss 1.930121
iteration 1100 / 3000: loss 1.996365
iteration 1200 / 3000: loss 1.863751
iteration 1300 / 3000: loss 1.865116
iteration 1400 / 3000: loss 1.905642
iteration 1500 / 3000: loss 1.890493
iteration 1600 / 3000: loss 1.854396
iteration 1700 / 3000: loss 1.773150
iteration 1800 / 3000: loss 1.841489
iteration 1900 / 3000: loss 1.918407
iteration 2000 / 3000: loss 1.811878
iteration 2100 / 3000: loss 1.711787
iteration 2200 / 3000: loss 1.800602
iteration 2300 / 3000: loss 1.717463
iteration 2400 / 3000: loss 1.792638
iteration 2500 / 3000: loss 1.756333
iteration 2600 / 3000: loss 1.687884
iteration 2700 / 3000: loss 1.821500
iteration 2800 / 3000: loss 1.758978
iteration 2900 / 3000: loss 1.731975
Validation accuracy:  0.398
```

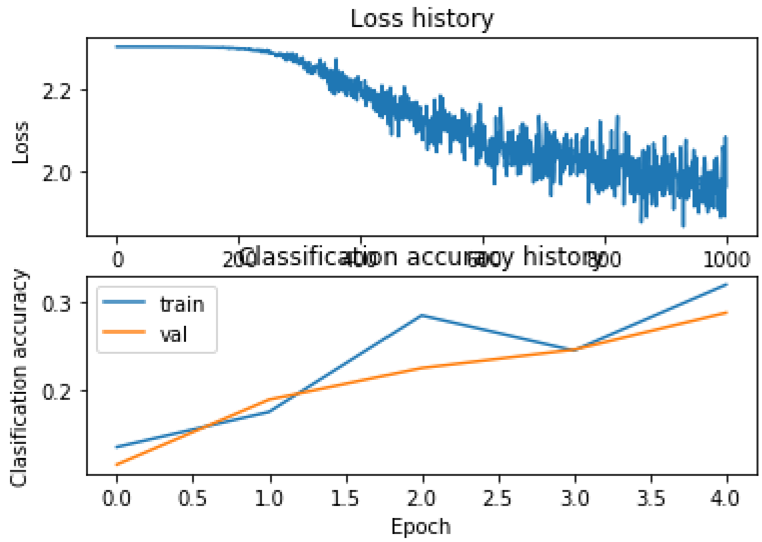
Debug the training

With the default parameters we provided above, you should get a validation accuracy of about 0.29 on the validation set. This isn't very good.

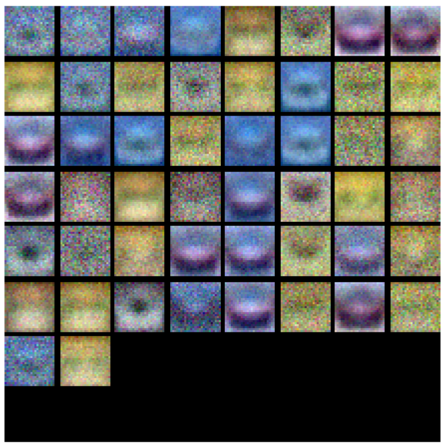
One strategy for getting insight into what's wrong is to plot the loss function and the accuracies on the training and validation sets during optimization.

Another strategy is to visualize the weights that were learned in the first layer of the network. In most neural networks trained on visual data, the first layer weights typically show some visible structure when visualized.

In [11]:
```
# Plot the loss function and train / validation accuracies
plt.subplot(2, 1, 1)
plt.plot(stats['loss_history'])
plt.title('Loss history')
plt.xlabel('Iteration')
plt.ylabel('Loss')

plt.subplot(2, 1, 2)
plt.plot(stats['train_acc_history'], label='train')
plt.plot(stats['val_acc_history'], label='val')
plt.title('Classification accuracy history')
plt.xlabel('Epoch')
plt.ylabel('Clasification accuracy')
plt.legend()
plt.show()
```



In [12]:
```
from cs682.vis_utils import visualize_grid

# Visualize the weights of the network

def show_net_weights(net):
    W1 = net.params['W1']
    W1 = W1.reshape(32, 32, 3, -1).transpose(3, 0, 1, 2)
    plt.imshow(visualize_grid(W1, padding=3).astype('uint8'))
    plt.gca().axis('off')
    plt.show()

show_net_weights(net)
```



Tune your hyperparameters

What's wrong?. Looking at the visualizations above, we see that the loss is decreasing more or less linearly, which seems to suggest that the learning rate may be too low. Moreover, there is no gap between the training and validation accuracy, suggesting that the model we used has low capacity, and that we should increase its size. On the other hand, with a very large model we would expect to see more overfitting, which would manifest itself as a very large gap between the training and validation accuracy.

Tuning. Tuning the hyperparameters and developing intuition for how they affect the final performance is a large part of using Neural Networks, so we want you to get a lot of practice. Below, you should experiment with different values of the various hyperparameters, including hidden layer size, learning rate, numer of training epochs, and regularization strength. You might also consider tuning the learning rate decay, but you should be able to get good performance using the default value.

Approximate results. You should be aim to achieve a classification accuracy of greater than 48% on the validation set. Our best network gets over 52% on the validation set.

Experiment: You goal in this exercise is to get as good of a result on CIFAR-10 as you can, with a fully-connected Neural Network. Feel free implement your own techniques (e.g. PCA to reduce dimensionality, or adding dropout, or adding features to the solver, etc.).

In [15]:
```
best_net = None # store the best model into this
input_size = 32 * 32 * 3
learning_rates = [1e-3, 3e-3]
regularization_strengths = [0.2, 0.5]
hiddenSizes = range(100,101)
num_classes = 10
results = {}
best_val = -1
best_lr = -1
best_reg = -1
best_hidden_size = -1
inner_iterations = 3000
batch_size = 200
learning_rate_decay=0.95

######################################################################
############
# TODO: Tune hyperparameters using the validation set. Store your bes
t trained  #
# model in best_net.
#
#
#
# To help debug your network, it may help to use visualizations simil
ar to the  #
# ones we used above; these visualizations will have significant qual
itative    #
# differences from the ones we saw above for the poorly tuned networ
k.         #
#
#
# Tweaking hyperparameters by hand can be fun, but you might find it
 useful to  #
# write code to sweep through possible combinations of hyperparameter
s          #
# automatically like we did on the previous exercises.
#
######################################################################
############
num_iters = 10
for it in range(num_iters):
    for jt in range(num_iters):
        for kt in range(len(hiddenSizes)):
            learning_rate = learning_rates[0] + it * ((learning_rates
[1] - learning_rates[0]) / num_iters)
            reg = regularization_strengths[0] + jt * ((regularization
_strengths[1] - regularization_strengths[0])/ num_iters)
            hidden_size = hiddenSizes[kt]

            net = TwoLayerNet(input_size, hidden_size, num_classes)
            # Train the network
            stats = net.train(X_train, y_train, X_val, y_val,
                        learning_rate, learning_rate_decay,
                        reg, inner_iterations, batch_size,verbose=Fal
se)
```

```
            y_train_pred = net.predict(X_train)
            y_val_pred = net.predict(X_val)
            training_accuracy = np.mean(y_train == y_train_pred)
            validation_accuracy = np.mean(y_val == y_val_pred)
            results[(learning_rate, reg, hidden_size)] = (training_ac
curacy, validation_accuracy)
            print('lr %e reg %e hidden %f train accuracy: %f val accu
racy: %f' % (
                        learning_rate, reg, hidden_size, training_acc
uracy, validation_accuracy))
            if validation_accuracy > best_val:
                best_val = validation_accuracy
                best_net = net
                best_lr = learning_rate
                best_reg = reg
                best_hidden_size = hidden_size

print('best validation accuracy achieved during cross-validation: lr
%e reg %e hidden %f val accuracy: %f, ' % (learning_rate, reg, hidden
_size, best_val))
#####################################################################
############
#                               END OF YOUR CODE
#
#####################################################################
############
```

```
lr 1.000000e-03 reg 2.000000e-01 hidden 100.000000 train accuracy: 0.
580367 val accuracy: 0.511000
lr 1.000000e-03 reg 2.300000e-01 hidden 100.000000 train accuracy: 0.
580776 val accuracy: 0.502000
lr 1.000000e-03 reg 2.600000e-01 hidden 100.000000 train accuracy: 0.
577776 val accuracy: 0.522000
lr 1.000000e-03 reg 2.900000e-01 hidden 100.000000 train accuracy: 0.
575959 val accuracy: 0.532000
lr 1.000000e-03 reg 3.200000e-01 hidden 100.000000 train accuracy: 0.
565367 val accuracy: 0.506000
lr 1.000000e-03 reg 3.500000e-01 hidden 100.000000 train accuracy: 0.
564122 val accuracy: 0.527000
lr 1.000000e-03 reg 3.800000e-01 hidden 100.000000 train accuracy: 0.
565306 val accuracy: 0.511000
lr 1.000000e-03 reg 4.100000e-01 hidden 100.000000 train accuracy: 0.
559551 val accuracy: 0.506000
lr 1.000000e-03 reg 4.400000e-01 hidden 100.000000 train accuracy: 0.
556673 val accuracy: 0.507000
lr 1.000000e-03 reg 4.700000e-01 hidden 100.000000 train accuracy: 0.
554041 val accuracy: 0.515000
lr 1.200000e-03 reg 2.000000e-01 hidden 100.000000 train accuracy: 0.
588694 val accuracy: 0.500000
lr 1.200000e-03 reg 2.300000e-01 hidden 100.000000 train accuracy: 0.
579306 val accuracy: 0.520000
lr 1.200000e-03 reg 2.600000e-01 hidden 100.000000 train accuracy: 0.
574980 val accuracy: 0.519000
lr 1.200000e-03 reg 2.900000e-01 hidden 100.000000 train accuracy: 0.
574633 val accuracy: 0.527000
lr 1.200000e-03 reg 3.200000e-01 hidden 100.000000 train accuracy: 0.
573122 val accuracy: 0.529000
lr 1.200000e-03 reg 3.500000e-01 hidden 100.000000 train accuracy: 0.
566327 val accuracy: 0.514000
lr 1.200000e-03 reg 3.800000e-01 hidden 100.000000 train accuracy: 0.
564653 val accuracy: 0.508000
lr 1.200000e-03 reg 4.100000e-01 hidden 100.000000 train accuracy: 0.
554449 val accuracy: 0.494000
lr 1.200000e-03 reg 4.400000e-01 hidden 100.000000 train accuracy: 0.
556388 val accuracy: 0.497000
lr 1.200000e-03 reg 4.700000e-01 hidden 100.000000 train accuracy: 0.
559306 val accuracy: 0.505000
lr 1.400000e-03 reg 2.000000e-01 hidden 100.000000 train accuracy: 0.
589898 val accuracy: 0.519000
lr 1.400000e-03 reg 2.300000e-01 hidden 100.000000 train accuracy: 0.
579367 val accuracy: 0.518000
lr 1.400000e-03 reg 2.600000e-01 hidden 100.000000 train accuracy: 0.
576204 val accuracy: 0.522000
lr 1.400000e-03 reg 2.900000e-01 hidden 100.000000 train accuracy: 0.
579429 val accuracy: 0.520000
lr 1.400000e-03 reg 3.200000e-01 hidden 100.000000 train accuracy: 0.
566388 val accuracy: 0.523000
lr 1.400000e-03 reg 3.500000e-01 hidden 100.000000 train accuracy: 0.
565694 val accuracy: 0.525000
lr 1.400000e-03 reg 3.800000e-01 hidden 100.000000 train accuracy: 0.
562306 val accuracy: 0.515000
lr 1.400000e-03 reg 4.100000e-01 hidden 100.000000 train accuracy: 0.
561918 val accuracy: 0.520000
lr 1.400000e-03 reg 4.400000e-01 hidden 100.000000 train accuracy: 0.
```

```
557939 val accuracy: 0.491000
lr 1.400000e-03 reg 4.700000e-01 hidden 100.000000 train accuracy: 0.
558286 val accuracy: 0.508000
lr 1.600000e-03 reg 2.000000e-01 hidden 100.000000 train accuracy: 0.
583571 val accuracy: 0.492000
lr 1.600000e-03 reg 2.300000e-01 hidden 100.000000 train accuracy: 0.
581898 val accuracy: 0.504000
lr 1.600000e-03 reg 2.600000e-01 hidden 100.000000 train accuracy: 0.
567286 val accuracy: 0.512000
lr 1.600000e-03 reg 2.900000e-01 hidden 100.000000 train accuracy: 0.
582857 val accuracy: 0.525000
lr 1.600000e-03 reg 3.200000e-01 hidden 100.000000 train accuracy: 0.
574816 val accuracy: 0.524000
lr 1.600000e-03 reg 3.500000e-01 hidden 100.000000 train accuracy: 0.
567429 val accuracy: 0.516000
lr 1.600000e-03 reg 3.800000e-01 hidden 100.000000 train accuracy: 0.
552041 val accuracy: 0.500000
lr 1.600000e-03 reg 4.100000e-01 hidden 100.000000 train accuracy: 0.
551143 val accuracy: 0.511000
lr 1.600000e-03 reg 4.400000e-01 hidden 100.000000 train accuracy: 0.
544633 val accuracy: 0.496000
lr 1.600000e-03 reg 4.700000e-01 hidden 100.000000 train accuracy: 0.
541673 val accuracy: 0.491000
lr 1.800000e-03 reg 2.000000e-01 hidden 100.000000 train accuracy: 0.
577694 val accuracy: 0.509000
lr 1.800000e-03 reg 2.300000e-01 hidden 100.000000 train accuracy: 0.
582490 val accuracy: 0.530000
lr 1.800000e-03 reg 2.600000e-01 hidden 100.000000 train accuracy: 0.
540673 val accuracy: 0.493000
lr 1.800000e-03 reg 2.900000e-01 hidden 100.000000 train accuracy: 0.
568980 val accuracy: 0.519000
lr 1.800000e-03 reg 3.200000e-01 hidden 100.000000 train accuracy: 0.
559653 val accuracy: 0.512000
lr 1.800000e-03 reg 3.500000e-01 hidden 100.000000 train accuracy: 0.
552347 val accuracy: 0.495000
lr 1.800000e-03 reg 3.800000e-01 hidden 100.000000 train accuracy: 0.
557245 val accuracy: 0.495000
lr 1.800000e-03 reg 4.100000e-01 hidden 100.000000 train accuracy: 0.
562939 val accuracy: 0.521000
lr 1.800000e-03 reg 4.400000e-01 hidden 100.000000 train accuracy: 0.
551286 val accuracy: 0.506000
lr 1.800000e-03 reg 4.700000e-01 hidden 100.000000 train accuracy: 0.
551878 val accuracy: 0.509000
lr 2.000000e-03 reg 2.000000e-01 hidden 100.000000 train accuracy: 0.
592306 val accuracy: 0.517000
lr 2.000000e-03 reg 2.300000e-01 hidden 100.000000 train accuracy: 0.
574776 val accuracy: 0.499000
lr 2.000000e-03 reg 2.600000e-01 hidden 100.000000 train accuracy: 0.
560857 val accuracy: 0.502000
lr 2.000000e-03 reg 2.900000e-01 hidden 100.000000 train accuracy: 0.
560694 val accuracy: 0.492000
lr 2.000000e-03 reg 3.200000e-01 hidden 100.000000 train accuracy: 0.
569959 val accuracy: 0.508000
lr 2.000000e-03 reg 3.500000e-01 hidden 100.000000 train accuracy: 0.
559408 val accuracy: 0.504000
lr 2.000000e-03 reg 3.800000e-01 hidden 100.000000 train accuracy: 0.
556245 val accuracy: 0.503000
```

```
lr 2.000000e-03 reg 4.100000e-01 hidden 100.000000 train accuracy: 0.
558673 val accuracy: 0.521000
lr 2.000000e-03 reg 4.400000e-01 hidden 100.000000 train accuracy: 0.
553184 val accuracy: 0.507000
lr 2.000000e-03 reg 4.700000e-01 hidden 100.000000 train accuracy: 0.
547592 val accuracy: 0.509000
lr 2.200000e-03 reg 2.000000e-01 hidden 100.000000 train accuracy: 0.
579143 val accuracy: 0.505000
lr 2.200000e-03 reg 2.300000e-01 hidden 100.000000 train accuracy: 0.
572122 val accuracy: 0.530000
lr 2.200000e-03 reg 2.600000e-01 hidden 100.000000 train accuracy: 0.
582816 val accuracy: 0.529000
lr 2.200000e-03 reg 2.900000e-01 hidden 100.000000 train accuracy: 0.
551776 val accuracy: 0.509000
lr 2.200000e-03 reg 3.200000e-01 hidden 100.000000 train accuracy: 0.
561694 val accuracy: 0.498000
lr 2.200000e-03 reg 3.500000e-01 hidden 100.000000 train accuracy: 0.
556612 val accuracy: 0.503000
lr 2.200000e-03 reg 3.800000e-01 hidden 100.000000 train accuracy: 0.
554408 val accuracy: 0.511000
lr 2.200000e-03 reg 4.100000e-01 hidden 100.000000 train accuracy: 0.
541571 val accuracy: 0.504000
lr 2.200000e-03 reg 4.400000e-01 hidden 100.000000 train accuracy: 0.
546673 val accuracy: 0.497000
lr 2.200000e-03 reg 4.700000e-01 hidden 100.000000 train accuracy: 0.
545633 val accuracy: 0.495000
lr 2.400000e-03 reg 2.000000e-01 hidden 100.000000 train accuracy: 0.
584041 val accuracy: 0.516000
lr 2.400000e-03 reg 2.300000e-01 hidden 100.000000 train accuracy: 0.
570592 val accuracy: 0.495000
lr 2.400000e-03 reg 2.600000e-01 hidden 100.000000 train accuracy: 0.
568612 val accuracy: 0.514000
lr 2.400000e-03 reg 2.900000e-01 hidden 100.000000 train accuracy: 0.
568020 val accuracy: 0.503000
lr 2.400000e-03 reg 3.200000e-01 hidden 100.000000 train accuracy: 0.
558714 val accuracy: 0.496000
lr 2.400000e-03 reg 3.500000e-01 hidden 100.000000 train accuracy: 0.
544776 val accuracy: 0.487000
lr 2.400000e-03 reg 3.800000e-01 hidden 100.000000 train accuracy: 0.
536061 val accuracy: 0.494000
lr 2.400000e-03 reg 4.100000e-01 hidden 100.000000 train accuracy: 0.
544571 val accuracy: 0.500000
lr 2.400000e-03 reg 4.400000e-01 hidden 100.000000 train accuracy: 0.
541776 val accuracy: 0.502000
lr 2.400000e-03 reg 4.700000e-01 hidden 100.000000 train accuracy: 0.
532510 val accuracy: 0.496000
lr 2.600000e-03 reg 2.000000e-01 hidden 100.000000 train accuracy: 0.
568633 val accuracy: 0.505000
lr 2.600000e-03 reg 2.300000e-01 hidden 100.000000 train accuracy: 0.
556959 val accuracy: 0.505000
lr 2.600000e-03 reg 2.600000e-01 hidden 100.000000 train accuracy: 0.
551429 val accuracy: 0.504000
lr 2.600000e-03 reg 2.900000e-01 hidden 100.000000 train accuracy: 0.
525796 val accuracy: 0.484000
lr 2.600000e-03 reg 3.200000e-01 hidden 100.000000 train accuracy: 0.
544041 val accuracy: 0.502000
lr 2.600000e-03 reg 3.500000e-01 hidden 100.000000 train accuracy: 0.
```

```
541735 val accuracy: 0.494000
lr 2.600000e-03 reg 3.800000e-01 hidden 100.000000 train accuracy: 0.
549735 val accuracy: 0.495000
lr 2.600000e-03 reg 4.100000e-01 hidden 100.000000 train accuracy: 0.
539551 val accuracy: 0.513000
lr 2.600000e-03 reg 4.400000e-01 hidden 100.000000 train accuracy: 0.
531857 val accuracy: 0.492000
lr 2.600000e-03 reg 4.700000e-01 hidden 100.000000 train accuracy: 0.
540857 val accuracy: 0.516000
lr 2.800000e-03 reg 2.000000e-01 hidden 100.000000 train accuracy: 0.
545755 val accuracy: 0.503000
lr 2.800000e-03 reg 2.300000e-01 hidden 100.000000 train accuracy: 0.
560449 val accuracy: 0.508000
lr 2.800000e-03 reg 2.600000e-01 hidden 100.000000 train accuracy: 0.
556102 val accuracy: 0.502000
lr 2.800000e-03 reg 2.900000e-01 hidden 100.000000 train accuracy: 0.
554327 val accuracy: 0.507000
lr 2.800000e-03 reg 3.200000e-01 hidden 100.000000 train accuracy: 0.
551388 val accuracy: 0.489000
lr 2.800000e-03 reg 3.500000e-01 hidden 100.000000 train accuracy: 0.
503163 val accuracy: 0.457000
lr 2.800000e-03 reg 3.800000e-01 hidden 100.000000 train accuracy: 0.
535837 val accuracy: 0.497000
lr 2.800000e-03 reg 4.100000e-01 hidden 100.000000 train accuracy: 0.
526571 val accuracy: 0.478000
lr 2.800000e-03 reg 4.400000e-01 hidden 100.000000 train accuracy: 0.
540592 val accuracy: 0.503000
lr 2.800000e-03 reg 4.700000e-01 hidden 100.000000 train accuracy: 0.
524082 val accuracy: 0.478000
best validation accuracy achieved during cross-validation: lr 2.80000
0e-03 reg 4.700000e-01 hidden 100.000000 val accuracy: 0.532000,
```

In [16]:
```
# visualize the weights of the best network
show_net_weights(best_net)
```

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%.

In [17]:
```
test_acc = (best_net.predict(X_test) == y_test).mean()
print('Test accuracy: ', test_acc)
```

```
Test accuracy:  0.519
```

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: 1,3

Your explanation:

1. This is TRUE because Training on a large dataset might make the model more generalized and might decrease the gap between test and trainign data accuracy.
2. This is FALSE because adding more hidden layers might overfit the training data more.
3. This is TRUE because increasing the regularization strength might handle the overfitting problem more effectively.

Image features exercise

Complete and hand in this completed worksheet (including its outputs and any supporting code outside of the worksheet) with your assignment submission. For more details see the [assignments page (https://compsci682-fa19.github.io/assignments2019/assignment1)](https://compsci682-fa19.github.io/assignments2019/assignment1) on the course website.

We have seen that we can achieve reasonable performance on an image classification task by training a linear classifier on the pixels of the input image. In this exercise we will show that we can improve our classification performance by training linear classifiers not on raw pixels but on features that are computed from the raw pixels.

All of your work for this exercise will be done in this notebook.

In [1]:
```
from __future__ import print_function
import random
import numpy as np
from cs682.data_utils import load_CIFAR10
import matplotlib.pyplot as plt


%matplotlib inline
plt.rcParams['figure.figsize'] = (10.0, 8.0) # set default size of pl
ots
plt.rcParams['image.interpolation'] = 'nearest'
plt.rcParams['image.cmap'] = 'gray'

# for auto-reloading extenrnal modules
# see http://stackoverflow.com/questions/1907993/autoreload-of-module
s-in-ipython
%load_ext autoreload
%autoreload 2
```

Load data

Similar to previous exercises, we will load CIFAR-10 data from disk.

In [2]:
```
from cs682.features import color_histogram_hsv, hog_feature

def get_CIFAR10_data(num_training=49000, num_validation=1000, num_tes
t=1000):
    # Load the raw CIFAR-10 data
    cifar10_dir = 'cs682/datasets/cifar-10-batches-py'

    X_train, y_train, X_test, y_test = load_CIFAR10(cifar10_dir)

    # Subsample the data
    mask = list(range(num_training, num_training + num_validation))
    X_val = X_train[mask]
    y_val = y_train[mask]
    mask = list(range(num_training))
    X_train = X_train[mask]
    y_train = y_train[mask]
    mask = list(range(num_test))
    X_test = X_test[mask]
    y_test = y_test[mask]

    return X_train, y_train, X_val, y_val, X_test, y_test

# Cleaning up variables to prevent loading data multiple times (which
may cause memory issue)
try:
   del X_train, y_train
   del X_test, y_test
   print('Clear previously loaded data.')
except:
   pass

X_train, y_train, X_val, y_val, X_test, y_test = get_CIFAR10_data()
```

Extract Features

For each image we will compute a Histogram of Oriented Gradients (HOG) as well as a color histogram using the hue channel in HSV color space. We form our final feature vector for each image by concatenating the HOG and color histogram feature vectors.

Roughly speaking, HOG should capture the texture of the image while ignoring color information, and the color histogram represents the color of the input image while ignoring texture. As a result, we expect that using both together ought to work better than using either alone. Verifying this assumption would be a good thing to try for your interests.

The `hog_feature` and `color_histogram_hsv` functions both operate on a single image and return a feature vector for that image. The extract_features function takes a set of images and a list of feature functions and evaluates each feature function on each image, storing the results in a matrix where each column is the concatenation of all feature vectors for a single image.

In [3]:
```
from cs682.features import *

num_color_bins = 10 # Number of bins in the color histogram
feature_fns = [hog_feature, lambda img: color_histogram_hsv(img, nbin
=num_color_bins)]
X_train_feats = extract_features(X_train, feature_fns, verbose=True)
X_val_feats = extract_features(X_val, feature_fns)
X_test_feats = extract_features(X_test, feature_fns)

# Preprocessing: Subtract the mean feature
mean_feat = np.mean(X_train_feats, axis=0, keepdims=True)
X_train_feats -= mean_feat
X_val_feats -= mean_feat
X_test_feats -= mean_feat

# Preprocessing: Divide by standard deviation. This ensures that each
feature
# has roughly the same scale.
std_feat = np.std(X_train_feats, axis=0, keepdims=True)
X_train_feats /= std_feat
X_val_feats /= std_feat
X_test_feats /= std_feat

# Preprocessing: Add a bias dimension
X_train_feats = np.hstack([X_train_feats, np.ones((X_train_feats.shap
e[0], 1))])
X_val_feats = np.hstack([X_val_feats, np.ones((X_val_feats.shape[0],
1))])
X_test_feats = np.hstack([X_test_feats, np.ones((X_test_feats.shape[0
], 1))])
```

```
Done extracting features for 1000 / 49000 images
Done extracting features for 2000 / 49000 images
Done extracting features for 3000 / 49000 images
Done extracting features for 4000 / 49000 images
Done extracting features for 5000 / 49000 images
Done extracting features for 6000 / 49000 images
Done extracting features for 7000 / 49000 images
Done extracting features for 8000 / 49000 images
Done extracting features for 9000 / 49000 images
Done extracting features for 10000 / 49000 images
Done extracting features for 11000 / 49000 images
Done extracting features for 12000 / 49000 images
Done extracting features for 13000 / 49000 images
Done extracting features for 14000 / 49000 images
Done extracting features for 15000 / 49000 images
Done extracting features for 16000 / 49000 images
Done extracting features for 17000 / 49000 images
Done extracting features for 18000 / 49000 images
Done extracting features for 19000 / 49000 images
Done extracting features for 20000 / 49000 images
Done extracting features for 21000 / 49000 images
Done extracting features for 22000 / 49000 images
Done extracting features for 23000 / 49000 images
Done extracting features for 24000 / 49000 images
Done extracting features for 25000 / 49000 images
Done extracting features for 26000 / 49000 images
Done extracting features for 27000 / 49000 images
Done extracting features for 28000 / 49000 images
Done extracting features for 29000 / 49000 images
Done extracting features for 30000 / 49000 images
Done extracting features for 31000 / 49000 images
Done extracting features for 32000 / 49000 images
Done extracting features for 33000 / 49000 images
Done extracting features for 34000 / 49000 images
Done extracting features for 35000 / 49000 images
Done extracting features for 36000 / 49000 images
Done extracting features for 37000 / 49000 images
Done extracting features for 38000 / 49000 images
Done extracting features for 39000 / 49000 images
Done extracting features for 40000 / 49000 images
Done extracting features for 41000 / 49000 images
Done extracting features for 42000 / 49000 images
Done extracting features for 43000 / 49000 images
Done extracting features for 44000 / 49000 images
Done extracting features for 45000 / 49000 images
Done extracting features for 46000 / 49000 images
Done extracting features for 47000 / 49000 images
Done extracting features for 48000 / 49000 images
```

Train SVM on features

Using the multiclass SVM code developed earlier in the assignment, train SVMs on top of the features extracted above; this should achieve better results than training SVMs directly on top of raw pixels.

In [4]:
```
# Use the validation set to tune the learning rate and regularization
strength

from cs682.classifiers.linear_classifier import LinearSVM

learning_rates = [1e-9, 1e-8, 1e-7]
regularization_strengths = [5e4, 5e5, 5e6]

results = {}
best_val = -1
best_svm = None

######################################################################
###########
# TODO:
#
# Use the validation set to set the learning rate and regularization
 strength. #
# This should be identical to the validation that you did for the SV
M; save    #
# the best trained classifer in best_svm. You might also want to play
#
# with different numbers of bins in the color histogram. If you are c
areful    #
# you should be able to get accuracy of near 0.44 on the validation s
et.       #
######################################################################
###########
learning_rates = [1e-9, 5e-5]
regularization_strengths = [2e4, 5e6]
num_iters = 20
for it in range(num_iters):
    for jt in range(num_iters):
        svm = LinearSVM()
        learning_rate = learning_rates[0] + it * ((learning_rates[1]
- learning_rates[0]) / num_iters)
        reg = regularization_strengths[0] + jt * ((regularization_str
engths[1] - regularization_strengths[0])/ num_iters)
        loss_hist = svm.train(X_train_feats, y_train, learning_rate=l
earning_rate, reg=reg,
                          num_iters=3000, verbose=False)
        y_train_pred = svm.predict(X_train_feats)
        y_val_pred = svm.predict(X_val_feats)
        training_accuracy = np.mean(y_train == y_train_pred)
        validation_accuracy = np.mean(y_val == y_val_pred)
        results[(learning_rate, reg)] = (training_accuracy, validatio
n_accuracy)
        print('lr %e reg %e  train accuracy: %f val accuracy: %f' % (
                    learning_rate, reg, training_accuracy, validation
_accuracy))
        if validation_accuracy > best_val:
            best_val = validation_accuracy
            best_svm = svm

######################################################################
###########
```

```
#                              END OF YOUR CODE
#
######################################################################
###########

# Print out results.
for lr, reg in sorted(results):
    train_accuracy, val_accuracy = results[(lr, reg)]
    print('lr %e reg %e train accuracy: %f val accuracy: %f' % (
                lr, reg, train_accuracy, val_accuracy))

print('best validation accuracy achieved during cross-validation: %f'
% best_val)
```

```
lr 1.000000e-09 reg 2.000000e+04  train accuracy: 0.096755 val accura
cy: 0.088000
lr 1.000000e-09 reg 2.690000e+05  train accuracy: 0.119918 val accura
cy: 0.098000
lr 1.000000e-09 reg 5.180000e+05  train accuracy: 0.085653 val accura
cy: 0.081000
lr 1.000000e-09 reg 7.670000e+05  train accuracy: 0.090796 val accura
cy: 0.090000
lr 1.000000e-09 reg 1.016000e+06  train accuracy: 0.098673 val accura
cy: 0.092000
lr 1.000000e-09 reg 1.265000e+06  train accuracy: 0.181918 val accura
cy: 0.163000
lr 1.000000e-09 reg 1.514000e+06  train accuracy: 0.331490 val accura
cy: 0.353000
lr 1.000000e-09 reg 1.763000e+06  train accuracy: 0.403327 val accura
cy: 0.398000
lr 1.000000e-09 reg 2.012000e+06  train accuracy: 0.416041 val accura
cy: 0.415000
lr 1.000000e-09 reg 2.261000e+06  train accuracy: 0.415449 val accura
cy: 0.417000
lr 1.000000e-09 reg 2.510000e+06  train accuracy: 0.413837 val accura
cy: 0.418000
lr 1.000000e-09 reg 2.759000e+06  train accuracy: 0.417020 val accura
cy: 0.423000
lr 1.000000e-09 reg 3.008000e+06  train accuracy: 0.412306 val accura
cy: 0.418000
lr 1.000000e-09 reg 3.257000e+06  train accuracy: 0.417163 val accura
cy: 0.416000
lr 1.000000e-09 reg 3.506000e+06  train accuracy: 0.417347 val accura
cy: 0.415000
lr 1.000000e-09 reg 3.755000e+06  train accuracy: 0.413531 val accura
cy: 0.418000
lr 1.000000e-09 reg 4.004000e+06  train accuracy: 0.413306 val accura
cy: 0.411000
lr 1.000000e-09 reg 4.253000e+06  train accuracy: 0.414204 val accura
cy: 0.419000
lr 1.000000e-09 reg 4.502000e+06  train accuracy: 0.416224 val accura
cy: 0.417000
lr 1.000000e-09 reg 4.751000e+06  train accuracy: 0.415939 val accura
cy: 0.416000
lr 2.500950e-06 reg 2.000000e+04  train accuracy: 0.397388 val accura
cy: 0.402000
lr 2.500950e-06 reg 2.690000e+05  train accuracy: 0.300714 val accura
cy: 0.292000
```

```
/home/nikhil/Desktop/git_workspace/Neural-Networks/assignment1/cs682/
classifiers/linear_svm.py:92: RuntimeWarning: overflow encountered in
double_scalars
  loss += reg * np.sum(W * W)
/home/nikhil/anaconda3/lib/python3.7/site-packages/numpy/core/fromnum
eric.py:86: RuntimeWarning: overflow encountered in reduce
  return ufunc.reduce(obj, axis, dtype, out, **passkwargs)
/home/nikhil/Desktop/git_workspace/Neural-Networks/assignment1/cs682/
classifiers/linear_svm.py:92: RuntimeWarning: overflow encountered in
multiply
  loss += reg * np.sum(W * W)
/home/nikhil/Desktop/git_workspace/Neural-Networks/assignment1/cs682/
classifiers/linear_svm.py:116: RuntimeWarning: overflow encountered i
n multiply
  dW += 2 * reg * W
/home/nikhil/Desktop/git_workspace/Neural-Networks/assignment1/cs682/
classifiers/linear_svm.py:87: RuntimeWarning: invalid value encounter
ed in subtract
  diffs = scores.T - actuals + 1 #(10*500)
/home/nikhil/Desktop/git_workspace/Neural-Networks/assignment1/cs682/
classifiers/linear_svm.py:88: RuntimeWarning: invalid value encounter
ed in less
  diffs[np.where(diffs < 0)] = 0
/home/nikhil/Desktop/git_workspace/Neural-Networks/assignment1/cs682/
classifiers/linear_svm.py:109: RuntimeWarning: invalid value encounte
red in greater
  diffs[np.where(diffs > 0)] = 1 #(10*500)
```

```
lr 2.500950e-06 reg 5.180000e+05  train accuracy: 0.100265 val accura
cy: 0.087000
lr 2.500950e-06 reg 7.670000e+05  train accuracy: 0.100265 val accura
cy: 0.087000
lr 2.500950e-06 reg 1.016000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 2.500950e-06 reg 1.265000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 2.500950e-06 reg 1.514000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 2.500950e-06 reg 1.763000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 2.500950e-06 reg 2.012000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 2.500950e-06 reg 2.261000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 2.500950e-06 reg 2.510000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 2.500950e-06 reg 2.759000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 2.500950e-06 reg 3.008000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 2.500950e-06 reg 3.257000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 2.500950e-06 reg 3.506000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 2.500950e-06 reg 3.755000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 2.500950e-06 reg 4.004000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 2.500950e-06 reg 4.253000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 2.500950e-06 reg 4.502000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 2.500950e-06 reg 4.751000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 5.000900e-06 reg 2.000000e+04  train accuracy: 0.392592 val accura
cy: 0.380000
lr 5.000900e-06 reg 2.690000e+05  train accuracy: 0.100265 val accura
cy: 0.087000
lr 5.000900e-06 reg 5.180000e+05  train accuracy: 0.100265 val accura
cy: 0.087000
lr 5.000900e-06 reg 7.670000e+05  train accuracy: 0.100265 val accura
cy: 0.087000
lr 5.000900e-06 reg 1.016000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 5.000900e-06 reg 1.265000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 5.000900e-06 reg 1.514000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 5.000900e-06 reg 1.763000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 5.000900e-06 reg 2.012000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 5.000900e-06 reg 2.261000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 5.000900e-06 reg 2.510000e+06  train accuracy: 0.100265 val accura
```

```
cy: 0.087000
lr 5.000900e-06 reg 2.759000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 5.000900e-06 reg 3.008000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 5.000900e-06 reg 3.257000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 5.000900e-06 reg 3.506000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 5.000900e-06 reg 3.755000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 5.000900e-06 reg 4.004000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 5.000900e-06 reg 4.253000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 5.000900e-06 reg 4.502000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 5.000900e-06 reg 4.751000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 7.500850e-06 reg 2.000000e+04  train accuracy: 0.403286 val accura
cy: 0.398000
lr 7.500850e-06 reg 2.690000e+05  train accuracy: 0.100265 val accura
cy: 0.087000
lr 7.500850e-06 reg 5.180000e+05  train accuracy: 0.100265 val accura
cy: 0.087000
lr 7.500850e-06 reg 7.670000e+05  train accuracy: 0.100265 val accura
cy: 0.087000
lr 7.500850e-06 reg 1.016000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 7.500850e-06 reg 1.265000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 7.500850e-06 reg 1.514000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 7.500850e-06 reg 1.763000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 7.500850e-06 reg 2.012000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 7.500850e-06 reg 2.261000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 7.500850e-06 reg 2.510000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 7.500850e-06 reg 2.759000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 7.500850e-06 reg 3.008000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 7.500850e-06 reg 3.257000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 7.500850e-06 reg 3.506000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 7.500850e-06 reg 3.755000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 7.500850e-06 reg 4.004000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 7.500850e-06 reg 4.253000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 7.500850e-06 reg 4.502000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
```

```
lr 7.500850e-06 reg 4.751000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 1.000080e-05 reg 2.000000e+04  train accuracy: 0.380245 val accura
cy: 0.383000
lr 1.000080e-05 reg 2.690000e+05  train accuracy: 0.100265 val accura
cy: 0.087000
lr 1.000080e-05 reg 5.180000e+05  train accuracy: 0.100265 val accura
cy: 0.087000
lr 1.000080e-05 reg 7.670000e+05  train accuracy: 0.100265 val accura
cy: 0.087000
lr 1.000080e-05 reg 1.016000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 1.000080e-05 reg 1.265000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 1.000080e-05 reg 1.514000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 1.000080e-05 reg 1.763000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 1.000080e-05 reg 2.012000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 1.000080e-05 reg 2.261000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 1.000080e-05 reg 2.510000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 1.000080e-05 reg 2.759000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 1.000080e-05 reg 3.008000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 1.000080e-05 reg 3.257000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 1.000080e-05 reg 3.506000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 1.000080e-05 reg 3.755000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 1.000080e-05 reg 4.004000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 1.000080e-05 reg 4.253000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 1.000080e-05 reg 4.502000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 1.000080e-05 reg 4.751000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 1.250075e-05 reg 2.000000e+04  train accuracy: 0.394245 val accura
cy: 0.406000
lr 1.250075e-05 reg 2.690000e+05  train accuracy: 0.100265 val accura
cy: 0.087000
lr 1.250075e-05 reg 5.180000e+05  train accuracy: 0.100265 val accura
cy: 0.087000
lr 1.250075e-05 reg 7.670000e+05  train accuracy: 0.100265 val accura
cy: 0.087000
lr 1.250075e-05 reg 1.016000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 1.250075e-05 reg 1.265000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 1.250075e-05 reg 1.514000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 1.250075e-05 reg 1.763000e+06  train accuracy: 0.100265 val accura
```

```
cy: 0.087000
lr 1.250075e-05 reg 2.012000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 1.250075e-05 reg 2.261000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 1.250075e-05 reg 2.510000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 1.250075e-05 reg 2.759000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 1.250075e-05 reg 3.008000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 1.250075e-05 reg 3.257000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 1.250075e-05 reg 3.506000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 1.250075e-05 reg 3.755000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 1.250075e-05 reg 4.004000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 1.250075e-05 reg 4.253000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 1.250075e-05 reg 4.502000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 1.250075e-05 reg 4.751000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 1.500070e-05 reg 2.000000e+04  train accuracy: 0.357061 val accura
cy: 0.349000
lr 1.500070e-05 reg 2.690000e+05  train accuracy: 0.100265 val accura
cy: 0.087000
lr 1.500070e-05 reg 5.180000e+05  train accuracy: 0.100265 val accura
cy: 0.087000
lr 1.500070e-05 reg 7.670000e+05  train accuracy: 0.100265 val accura
cy: 0.087000
lr 1.500070e-05 reg 1.016000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 1.500070e-05 reg 1.265000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 1.500070e-05 reg 1.514000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 1.500070e-05 reg 1.763000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 1.500070e-05 reg 2.012000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 1.500070e-05 reg 2.261000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 1.500070e-05 reg 2.510000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 1.500070e-05 reg 2.759000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 1.500070e-05 reg 3.008000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 1.500070e-05 reg 3.257000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 1.500070e-05 reg 3.506000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 1.500070e-05 reg 3.755000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
```

```
lr 1.500070e-05 reg 4.004000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 1.500070e-05 reg 4.253000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 1.500070e-05 reg 4.502000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 1.500070e-05 reg 4.751000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 1.750065e-05 reg 2.000000e+04  train accuracy: 0.342980 val accura
cy: 0.357000
lr 1.750065e-05 reg 2.690000e+05  train accuracy: 0.100265 val accura
cy: 0.087000
lr 1.750065e-05 reg 5.180000e+05  train accuracy: 0.100265 val accura
cy: 0.087000
lr 1.750065e-05 reg 7.670000e+05  train accuracy: 0.100265 val accura
cy: 0.087000
lr 1.750065e-05 reg 1.016000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 1.750065e-05 reg 1.265000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 1.750065e-05 reg 1.514000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 1.750065e-05 reg 1.763000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 1.750065e-05 reg 2.012000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 1.750065e-05 reg 2.261000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 1.750065e-05 reg 2.510000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 1.750065e-05 reg 2.759000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 1.750065e-05 reg 3.008000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 1.750065e-05 reg 3.257000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 1.750065e-05 reg 3.506000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 1.750065e-05 reg 3.755000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 1.750065e-05 reg 4.004000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 1.750065e-05 reg 4.253000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 1.750065e-05 reg 4.502000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 1.750065e-05 reg 4.751000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 2.000060e-05 reg 2.000000e+04  train accuracy: 0.356714 val accura
cy: 0.371000
lr 2.000060e-05 reg 2.690000e+05  train accuracy: 0.100265 val accura
cy: 0.087000
lr 2.000060e-05 reg 5.180000e+05  train accuracy: 0.100265 val accura
cy: 0.087000
lr 2.000060e-05 reg 7.670000e+05  train accuracy: 0.100265 val accura
cy: 0.087000
lr 2.000060e-05 reg 1.016000e+06  train accuracy: 0.100265 val accura
```

```
cy: 0.087000
lr 2.000060e-05 reg 1.265000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 2.000060e-05 reg 1.514000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 2.000060e-05 reg 1.763000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 2.000060e-05 reg 2.012000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 2.000060e-05 reg 2.261000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 2.000060e-05 reg 2.510000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 2.000060e-05 reg 2.759000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 2.000060e-05 reg 3.008000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 2.000060e-05 reg 3.257000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 2.000060e-05 reg 3.506000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 2.000060e-05 reg 3.755000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 2.000060e-05 reg 4.004000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 2.000060e-05 reg 4.253000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 2.000060e-05 reg 4.502000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 2.000060e-05 reg 4.751000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 2.250055e-05 reg 2.000000e+04  train accuracy: 0.334122 val accura
cy: 0.333000
lr 2.250055e-05 reg 2.690000e+05  train accuracy: 0.100265 val accura
cy: 0.087000
lr 2.250055e-05 reg 5.180000e+05  train accuracy: 0.100265 val accura
cy: 0.087000
lr 2.250055e-05 reg 7.670000e+05  train accuracy: 0.100265 val accura
cy: 0.087000
lr 2.250055e-05 reg 1.016000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 2.250055e-05 reg 1.265000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 2.250055e-05 reg 1.514000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 2.250055e-05 reg 1.763000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 2.250055e-05 reg 2.012000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 2.250055e-05 reg 2.261000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 2.250055e-05 reg 2.510000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 2.250055e-05 reg 2.759000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 2.250055e-05 reg 3.008000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
```

```
lr 2.250055e-05 reg 3.257000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 2.250055e-05 reg 3.506000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 2.250055e-05 reg 3.755000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 2.250055e-05 reg 4.004000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 2.250055e-05 reg 4.253000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 2.250055e-05 reg 4.502000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 2.250055e-05 reg 4.751000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 2.500050e-05 reg 2.000000e+04  train accuracy: 0.341510 val accura
cy: 0.346000
lr 2.500050e-05 reg 2.690000e+05  train accuracy: 0.100265 val accura
cy: 0.087000
lr 2.500050e-05 reg 5.180000e+05  train accuracy: 0.100265 val accura
cy: 0.087000
lr 2.500050e-05 reg 7.670000e+05  train accuracy: 0.100265 val accura
cy: 0.087000
lr 2.500050e-05 reg 1.016000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 2.500050e-05 reg 1.265000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 2.500050e-05 reg 1.514000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 2.500050e-05 reg 1.763000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 2.500050e-05 reg 2.012000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 2.500050e-05 reg 2.261000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 2.500050e-05 reg 2.510000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 2.500050e-05 reg 2.759000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 2.500050e-05 reg 3.008000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 2.500050e-05 reg 3.257000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 2.500050e-05 reg 3.506000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 2.500050e-05 reg 3.755000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 2.500050e-05 reg 4.004000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 2.500050e-05 reg 4.253000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 2.500050e-05 reg 4.502000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 2.500050e-05 reg 4.751000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 2.750045e-05 reg 2.000000e+04  train accuracy: 0.322306 val accura
cy: 0.332000
lr 2.750045e-05 reg 2.690000e+05  train accuracy: 0.100265 val accura
```

```
cy: 0.087000
lr 2.750045e-05 reg 5.180000e+05  train accuracy: 0.100265 val accura
cy: 0.087000
lr 2.750045e-05 reg 7.670000e+05  train accuracy: 0.100265 val accura
cy: 0.087000
lr 2.750045e-05 reg 1.016000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 2.750045e-05 reg 1.265000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 2.750045e-05 reg 1.514000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 2.750045e-05 reg 1.763000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 2.750045e-05 reg 2.012000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 2.750045e-05 reg 2.261000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 2.750045e-05 reg 2.510000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 2.750045e-05 reg 2.759000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 2.750045e-05 reg 3.008000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 2.750045e-05 reg 3.257000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 2.750045e-05 reg 3.506000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 2.750045e-05 reg 3.755000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 2.750045e-05 reg 4.004000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 2.750045e-05 reg 4.253000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 2.750045e-05 reg 4.502000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 2.750045e-05 reg 4.751000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 3.000040e-05 reg 2.000000e+04  train accuracy: 0.310265 val accura
cy: 0.332000
lr 3.000040e-05 reg 2.690000e+05  train accuracy: 0.100265 val accura
cy: 0.087000
lr 3.000040e-05 reg 5.180000e+05  train accuracy: 0.100265 val accura
cy: 0.087000
lr 3.000040e-05 reg 7.670000e+05  train accuracy: 0.100265 val accura
cy: 0.087000
lr 3.000040e-05 reg 1.016000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 3.000040e-05 reg 1.265000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 3.000040e-05 reg 1.514000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 3.000040e-05 reg 1.763000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 3.000040e-05 reg 2.012000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 3.000040e-05 reg 2.261000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
```

```
lr 3.000040e-05 reg 2.510000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 3.000040e-05 reg 2.759000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 3.000040e-05 reg 3.008000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 3.000040e-05 reg 3.257000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 3.000040e-05 reg 3.506000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 3.000040e-05 reg 3.755000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 3.000040e-05 reg 4.004000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 3.000040e-05 reg 4.253000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 3.000040e-05 reg 4.502000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 3.000040e-05 reg 4.751000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 3.250035e-05 reg 2.000000e+04  train accuracy: 0.323449 val accura
cy: 0.317000
lr 3.250035e-05 reg 2.690000e+05  train accuracy: 0.100265 val accura
cy: 0.087000
lr 3.250035e-05 reg 5.180000e+05  train accuracy: 0.100265 val accura
cy: 0.087000
lr 3.250035e-05 reg 7.670000e+05  train accuracy: 0.100265 val accura
cy: 0.087000
lr 3.250035e-05 reg 1.016000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 3.250035e-05 reg 1.265000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 3.250035e-05 reg 1.514000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 3.250035e-05 reg 1.763000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 3.250035e-05 reg 2.012000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 3.250035e-05 reg 2.261000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 3.250035e-05 reg 2.510000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 3.250035e-05 reg 2.759000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 3.250035e-05 reg 3.008000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 3.250035e-05 reg 3.257000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 3.250035e-05 reg 3.506000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 3.250035e-05 reg 3.755000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 3.250035e-05 reg 4.004000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 3.250035e-05 reg 4.253000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 3.250035e-05 reg 4.502000e+06  train accuracy: 0.100265 val accura
```

```
cy: 0.087000
lr 3.250035e-05 reg 4.751000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 3.500030e-05 reg 2.000000e+04  train accuracy: 0.293082 val accura
cy: 0.326000
lr 3.500030e-05 reg 2.690000e+05  train accuracy: 0.100265 val accura
cy: 0.087000
lr 3.500030e-05 reg 5.180000e+05  train accuracy: 0.100265 val accura
cy: 0.087000
lr 3.500030e-05 reg 7.670000e+05  train accuracy: 0.100265 val accura
cy: 0.087000
lr 3.500030e-05 reg 1.016000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 3.500030e-05 reg 1.265000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 3.500030e-05 reg 1.514000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 3.500030e-05 reg 1.763000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 3.500030e-05 reg 2.012000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 3.500030e-05 reg 2.261000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 3.500030e-05 reg 2.510000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 3.500030e-05 reg 2.759000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 3.500030e-05 reg 3.008000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 3.500030e-05 reg 3.257000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 3.500030e-05 reg 3.506000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 3.500030e-05 reg 3.755000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 3.500030e-05 reg 4.004000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 3.500030e-05 reg 4.253000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 3.500030e-05 reg 4.502000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 3.500030e-05 reg 4.751000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 3.750025e-05 reg 2.000000e+04  train accuracy: 0.288245 val accura
cy: 0.309000
lr 3.750025e-05 reg 2.690000e+05  train accuracy: 0.100265 val accura
cy: 0.087000
lr 3.750025e-05 reg 5.180000e+05  train accuracy: 0.100265 val accura
cy: 0.087000
lr 3.750025e-05 reg 7.670000e+05  train accuracy: 0.100265 val accura
cy: 0.087000
lr 3.750025e-05 reg 1.016000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 3.750025e-05 reg 1.265000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 3.750025e-05 reg 1.514000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
```

```
lr 3.750025e-05 reg 1.763000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 3.750025e-05 reg 2.012000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 3.750025e-05 reg 2.261000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 3.750025e-05 reg 2.510000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 3.750025e-05 reg 2.759000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 3.750025e-05 reg 3.008000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 3.750025e-05 reg 3.257000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 3.750025e-05 reg 3.506000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 3.750025e-05 reg 3.755000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 3.750025e-05 reg 4.004000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 3.750025e-05 reg 4.253000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 3.750025e-05 reg 4.502000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 3.750025e-05 reg 4.751000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 4.000020e-05 reg 2.000000e+04  train accuracy: 0.259469 val accura
cy: 0.258000
lr 4.000020e-05 reg 2.690000e+05  train accuracy: 0.100265 val accura
cy: 0.087000
lr 4.000020e-05 reg 5.180000e+05  train accuracy: 0.100265 val accura
cy: 0.087000
lr 4.000020e-05 reg 7.670000e+05  train accuracy: 0.100265 val accura
cy: 0.087000
lr 4.000020e-05 reg 1.016000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 4.000020e-05 reg 1.265000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 4.000020e-05 reg 1.514000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 4.000020e-05 reg 1.763000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 4.000020e-05 reg 2.012000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 4.000020e-05 reg 2.261000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 4.000020e-05 reg 2.510000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 4.000020e-05 reg 2.759000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 4.000020e-05 reg 3.008000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 4.000020e-05 reg 3.257000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 4.000020e-05 reg 3.506000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 4.000020e-05 reg 3.755000e+06  train accuracy: 0.100265 val accura
```

```
cy: 0.087000
lr 4.000020e-05 reg 4.004000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 4.000020e-05 reg 4.253000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 4.000020e-05 reg 4.502000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 4.000020e-05 reg 4.751000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 4.250015e-05 reg 2.000000e+04  train accuracy: 0.265082 val accura
cy: 0.277000
lr 4.250015e-05 reg 2.690000e+05  train accuracy: 0.100265 val accura
cy: 0.087000
lr 4.250015e-05 reg 5.180000e+05  train accuracy: 0.100265 val accura
cy: 0.087000
lr 4.250015e-05 reg 7.670000e+05  train accuracy: 0.100265 val accura
cy: 0.087000
lr 4.250015e-05 reg 1.016000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 4.250015e-05 reg 1.265000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 4.250015e-05 reg 1.514000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 4.250015e-05 reg 1.763000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 4.250015e-05 reg 2.012000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 4.250015e-05 reg 2.261000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 4.250015e-05 reg 2.510000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 4.250015e-05 reg 2.759000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 4.250015e-05 reg 3.008000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 4.250015e-05 reg 3.257000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 4.250015e-05 reg 3.506000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 4.250015e-05 reg 3.755000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 4.250015e-05 reg 4.004000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 4.250015e-05 reg 4.253000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 4.250015e-05 reg 4.502000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 4.250015e-05 reg 4.751000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 4.500010e-05 reg 2.000000e+04  train accuracy: 0.241735 val accura
cy: 0.260000
lr 4.500010e-05 reg 2.690000e+05  train accuracy: 0.100265 val accura
cy: 0.087000
lr 4.500010e-05 reg 5.180000e+05  train accuracy: 0.100265 val accura
cy: 0.087000
lr 4.500010e-05 reg 7.670000e+05  train accuracy: 0.100265 val accura
cy: 0.087000
```

```
lr 4.500010e-05 reg 1.016000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 4.500010e-05 reg 1.265000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 4.500010e-05 reg 1.514000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 4.500010e-05 reg 1.763000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 4.500010e-05 reg 2.012000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 4.500010e-05 reg 2.261000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 4.500010e-05 reg 2.510000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 4.500010e-05 reg 2.759000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 4.500010e-05 reg 3.008000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 4.500010e-05 reg 3.257000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 4.500010e-05 reg 3.506000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 4.500010e-05 reg 3.755000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 4.500010e-05 reg 4.004000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 4.500010e-05 reg 4.253000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 4.500010e-05 reg 4.502000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 4.500010e-05 reg 4.751000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 4.750005e-05 reg 2.000000e+04  train accuracy: 0.225286 val accura
cy: 0.252000
lr 4.750005e-05 reg 2.690000e+05  train accuracy: 0.100265 val accura
cy: 0.087000
lr 4.750005e-05 reg 5.180000e+05  train accuracy: 0.100265 val accura
cy: 0.087000
lr 4.750005e-05 reg 7.670000e+05  train accuracy: 0.100265 val accura
cy: 0.087000
lr 4.750005e-05 reg 1.016000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 4.750005e-05 reg 1.265000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 4.750005e-05 reg 1.514000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 4.750005e-05 reg 1.763000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 4.750005e-05 reg 2.012000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 4.750005e-05 reg 2.261000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 4.750005e-05 reg 2.510000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 4.750005e-05 reg 2.759000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 4.750005e-05 reg 3.008000e+06  train accuracy: 0.100265 val accura
```

```
cy: 0.087000
lr 4.750005e-05 reg 3.257000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 4.750005e-05 reg 3.506000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 4.750005e-05 reg 3.755000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 4.750005e-05 reg 4.004000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 4.750005e-05 reg 4.253000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 4.750005e-05 reg 4.502000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 4.750005e-05 reg 4.751000e+06  train accuracy: 0.100265 val accura
cy: 0.087000
lr 1.000000e-09 reg 2.000000e+04 train accuracy: 0.096755 val accurac
y: 0.088000
lr 1.000000e-09 reg 2.690000e+05 train accuracy: 0.119918 val accurac
y: 0.098000
lr 1.000000e-09 reg 5.180000e+05 train accuracy: 0.085653 val accurac
y: 0.081000
lr 1.000000e-09 reg 7.670000e+05 train accuracy: 0.090796 val accurac
y: 0.090000
lr 1.000000e-09 reg 1.016000e+06 train accuracy: 0.098673 val accurac
y: 0.092000
lr 1.000000e-09 reg 1.265000e+06 train accuracy: 0.181918 val accurac
y: 0.163000
lr 1.000000e-09 reg 1.514000e+06 train accuracy: 0.331490 val accurac
y: 0.353000
lr 1.000000e-09 reg 1.763000e+06 train accuracy: 0.403327 val accurac
y: 0.398000
lr 1.000000e-09 reg 2.012000e+06 train accuracy: 0.416041 val accurac
y: 0.415000
lr 1.000000e-09 reg 2.261000e+06 train accuracy: 0.415449 val accurac
y: 0.417000
lr 1.000000e-09 reg 2.510000e+06 train accuracy: 0.413837 val accurac
y: 0.418000
lr 1.000000e-09 reg 2.759000e+06 train accuracy: 0.417020 val accurac
y: 0.423000
lr 1.000000e-09 reg 3.008000e+06 train accuracy: 0.412306 val accurac
y: 0.418000
lr 1.000000e-09 reg 3.257000e+06 train accuracy: 0.417163 val accurac
y: 0.416000
lr 1.000000e-09 reg 3.506000e+06 train accuracy: 0.417347 val accurac
y: 0.415000
lr 1.000000e-09 reg 3.755000e+06 train accuracy: 0.413531 val accurac
y: 0.418000
lr 1.000000e-09 reg 4.004000e+06 train accuracy: 0.413306 val accurac
y: 0.411000
lr 1.000000e-09 reg 4.253000e+06 train accuracy: 0.414204 val accurac
y: 0.419000
lr 1.000000e-09 reg 4.502000e+06 train accuracy: 0.416224 val accurac
y: 0.417000
lr 1.000000e-09 reg 4.751000e+06 train accuracy: 0.415939 val accurac
y: 0.416000
lr 2.500950e-06 reg 2.000000e+04 train accuracy: 0.397388 val accurac
y: 0.402000
```

```
lr 2.500950e-06 reg 2.690000e+05 train accuracy: 0.300714 val accurac
y: 0.292000
lr 2.500950e-06 reg 5.180000e+05 train accuracy: 0.100265 val accurac
y: 0.087000
lr 2.500950e-06 reg 7.670000e+05 train accuracy: 0.100265 val accurac
y: 0.087000
lr 2.500950e-06 reg 1.016000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 2.500950e-06 reg 1.265000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 2.500950e-06 reg 1.514000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 2.500950e-06 reg 1.763000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 2.500950e-06 reg 2.012000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 2.500950e-06 reg 2.261000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 2.500950e-06 reg 2.510000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 2.500950e-06 reg 2.759000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 2.500950e-06 reg 3.008000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 2.500950e-06 reg 3.257000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 2.500950e-06 reg 3.506000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 2.500950e-06 reg 3.755000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 2.500950e-06 reg 4.004000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 2.500950e-06 reg 4.253000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 2.500950e-06 reg 4.502000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 2.500950e-06 reg 4.751000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 5.000900e-06 reg 2.000000e+04 train accuracy: 0.392592 val accurac
y: 0.380000
lr 5.000900e-06 reg 2.690000e+05 train accuracy: 0.100265 val accurac
y: 0.087000
lr 5.000900e-06 reg 5.180000e+05 train accuracy: 0.100265 val accurac
y: 0.087000
lr 5.000900e-06 reg 7.670000e+05 train accuracy: 0.100265 val accurac
y: 0.087000
lr 5.000900e-06 reg 1.016000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 5.000900e-06 reg 1.265000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 5.000900e-06 reg 1.514000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 5.000900e-06 reg 1.763000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 5.000900e-06 reg 2.012000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 5.000900e-06 reg 2.261000e+06 train accuracy: 0.100265 val accurac
```

```
y: 0.087000
lr 5.000900e-06 reg 2.510000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 5.000900e-06 reg 2.759000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 5.000900e-06 reg 3.008000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 5.000900e-06 reg 3.257000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 5.000900e-06 reg 3.506000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 5.000900e-06 reg 3.755000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 5.000900e-06 reg 4.004000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 5.000900e-06 reg 4.253000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 5.000900e-06 reg 4.502000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 5.000900e-06 reg 4.751000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 7.500850e-06 reg 2.000000e+04 train accuracy: 0.403286 val accurac
y: 0.398000
lr 7.500850e-06 reg 2.690000e+05 train accuracy: 0.100265 val accurac
y: 0.087000
lr 7.500850e-06 reg 5.180000e+05 train accuracy: 0.100265 val accurac
y: 0.087000
lr 7.500850e-06 reg 7.670000e+05 train accuracy: 0.100265 val accurac
y: 0.087000
lr 7.500850e-06 reg 1.016000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 7.500850e-06 reg 1.265000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 7.500850e-06 reg 1.514000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 7.500850e-06 reg 1.763000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 7.500850e-06 reg 2.012000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 7.500850e-06 reg 2.261000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 7.500850e-06 reg 2.510000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 7.500850e-06 reg 2.759000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 7.500850e-06 reg 3.008000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 7.500850e-06 reg 3.257000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 7.500850e-06 reg 3.506000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 7.500850e-06 reg 3.755000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 7.500850e-06 reg 4.004000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 7.500850e-06 reg 4.253000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
```

```
lr 7.500850e-06 reg 4.502000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 7.500850e-06 reg 4.751000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 1.000080e-05 reg 2.000000e+04 train accuracy: 0.380245 val accurac
y: 0.383000
lr 1.000080e-05 reg 2.690000e+05 train accuracy: 0.100265 val accurac
y: 0.087000
lr 1.000080e-05 reg 5.180000e+05 train accuracy: 0.100265 val accurac
y: 0.087000
lr 1.000080e-05 reg 7.670000e+05 train accuracy: 0.100265 val accurac
y: 0.087000
lr 1.000080e-05 reg 1.016000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 1.000080e-05 reg 1.265000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 1.000080e-05 reg 1.514000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 1.000080e-05 reg 1.763000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 1.000080e-05 reg 2.012000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 1.000080e-05 reg 2.261000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 1.000080e-05 reg 2.510000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 1.000080e-05 reg 2.759000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 1.000080e-05 reg 3.008000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 1.000080e-05 reg 3.257000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 1.000080e-05 reg 3.506000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 1.000080e-05 reg 3.755000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 1.000080e-05 reg 4.004000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 1.000080e-05 reg 4.253000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 1.000080e-05 reg 4.502000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 1.000080e-05 reg 4.751000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 1.250075e-05 reg 2.000000e+04 train accuracy: 0.394245 val accurac
y: 0.406000
lr 1.250075e-05 reg 2.690000e+05 train accuracy: 0.100265 val accurac
y: 0.087000
lr 1.250075e-05 reg 5.180000e+05 train accuracy: 0.100265 val accurac
y: 0.087000
lr 1.250075e-05 reg 7.670000e+05 train accuracy: 0.100265 val accurac
y: 0.087000
lr 1.250075e-05 reg 1.016000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 1.250075e-05 reg 1.265000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 1.250075e-05 reg 1.514000e+06 train accuracy: 0.100265 val accurac
```

```
y: 0.087000
lr 1.250075e-05 reg 1.763000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 1.250075e-05 reg 2.012000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 1.250075e-05 reg 2.261000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 1.250075e-05 reg 2.510000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 1.250075e-05 reg 2.759000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 1.250075e-05 reg 3.008000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 1.250075e-05 reg 3.257000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 1.250075e-05 reg 3.506000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 1.250075e-05 reg 3.755000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 1.250075e-05 reg 4.004000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 1.250075e-05 reg 4.253000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 1.250075e-05 reg 4.502000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 1.250075e-05 reg 4.751000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 1.500070e-05 reg 2.000000e+04 train accuracy: 0.357061 val accurac
y: 0.349000
lr 1.500070e-05 reg 2.690000e+05 train accuracy: 0.100265 val accurac
y: 0.087000
lr 1.500070e-05 reg 5.180000e+05 train accuracy: 0.100265 val accurac
y: 0.087000
lr 1.500070e-05 reg 7.670000e+05 train accuracy: 0.100265 val accurac
y: 0.087000
lr 1.500070e-05 reg 1.016000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 1.500070e-05 reg 1.265000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 1.500070e-05 reg 1.514000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 1.500070e-05 reg 1.763000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 1.500070e-05 reg 2.012000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 1.500070e-05 reg 2.261000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 1.500070e-05 reg 2.510000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 1.500070e-05 reg 2.759000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 1.500070e-05 reg 3.008000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 1.500070e-05 reg 3.257000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 1.500070e-05 reg 3.506000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
```

```
lr 1.500070e-05 reg 3.755000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 1.500070e-05 reg 4.004000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 1.500070e-05 reg 4.253000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 1.500070e-05 reg 4.502000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 1.500070e-05 reg 4.751000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 1.750065e-05 reg 2.000000e+04 train accuracy: 0.342980 val accurac
y: 0.357000
lr 1.750065e-05 reg 2.690000e+05 train accuracy: 0.100265 val accurac
y: 0.087000
lr 1.750065e-05 reg 5.180000e+05 train accuracy: 0.100265 val accurac
y: 0.087000
lr 1.750065e-05 reg 7.670000e+05 train accuracy: 0.100265 val accurac
y: 0.087000
lr 1.750065e-05 reg 1.016000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 1.750065e-05 reg 1.265000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 1.750065e-05 reg 1.514000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 1.750065e-05 reg 1.763000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 1.750065e-05 reg 2.012000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 1.750065e-05 reg 2.261000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 1.750065e-05 reg 2.510000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 1.750065e-05 reg 2.759000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 1.750065e-05 reg 3.008000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 1.750065e-05 reg 3.257000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 1.750065e-05 reg 3.506000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 1.750065e-05 reg 3.755000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 1.750065e-05 reg 4.004000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 1.750065e-05 reg 4.253000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 1.750065e-05 reg 4.502000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 1.750065e-05 reg 4.751000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 2.000060e-05 reg 2.000000e+04 train accuracy: 0.356714 val accurac
y: 0.371000
lr 2.000060e-05 reg 2.690000e+05 train accuracy: 0.100265 val accurac
y: 0.087000
lr 2.000060e-05 reg 5.180000e+05 train accuracy: 0.100265 val accurac
y: 0.087000
lr 2.000060e-05 reg 7.670000e+05 train accuracy: 0.100265 val accurac
```

```
y: 0.087000
lr 2.000060e-05 reg 1.016000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 2.000060e-05 reg 1.265000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 2.000060e-05 reg 1.514000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 2.000060e-05 reg 1.763000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 2.000060e-05 reg 2.012000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 2.000060e-05 reg 2.261000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 2.000060e-05 reg 2.510000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 2.000060e-05 reg 2.759000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 2.000060e-05 reg 3.008000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 2.000060e-05 reg 3.257000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 2.000060e-05 reg 3.506000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 2.000060e-05 reg 3.755000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 2.000060e-05 reg 4.004000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 2.000060e-05 reg 4.253000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 2.000060e-05 reg 4.502000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 2.000060e-05 reg 4.751000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 2.250055e-05 reg 2.000000e+04 train accuracy: 0.334122 val accurac
y: 0.333000
lr 2.250055e-05 reg 2.690000e+05 train accuracy: 0.100265 val accurac
y: 0.087000
lr 2.250055e-05 reg 5.180000e+05 train accuracy: 0.100265 val accurac
y: 0.087000
lr 2.250055e-05 reg 7.670000e+05 train accuracy: 0.100265 val accurac
y: 0.087000
lr 2.250055e-05 reg 1.016000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 2.250055e-05 reg 1.265000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 2.250055e-05 reg 1.514000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 2.250055e-05 reg 1.763000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 2.250055e-05 reg 2.012000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 2.250055e-05 reg 2.261000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 2.250055e-05 reg 2.510000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 2.250055e-05 reg 2.759000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
```

```
lr 2.250055e-05 reg 3.008000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 2.250055e-05 reg 3.257000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 2.250055e-05 reg 3.506000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 2.250055e-05 reg 3.755000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 2.250055e-05 reg 4.004000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 2.250055e-05 reg 4.253000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 2.250055e-05 reg 4.502000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 2.250055e-05 reg 4.751000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 2.500050e-05 reg 2.000000e+04 train accuracy: 0.341510 val accurac
y: 0.346000
lr 2.500050e-05 reg 2.690000e+05 train accuracy: 0.100265 val accurac
y: 0.087000
lr 2.500050e-05 reg 5.180000e+05 train accuracy: 0.100265 val accurac
y: 0.087000
lr 2.500050e-05 reg 7.670000e+05 train accuracy: 0.100265 val accurac
y: 0.087000
lr 2.500050e-05 reg 1.016000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 2.500050e-05 reg 1.265000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 2.500050e-05 reg 1.514000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 2.500050e-05 reg 1.763000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 2.500050e-05 reg 2.012000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 2.500050e-05 reg 2.261000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 2.500050e-05 reg 2.510000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 2.500050e-05 reg 2.759000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 2.500050e-05 reg 3.008000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 2.500050e-05 reg 3.257000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 2.500050e-05 reg 3.506000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 2.500050e-05 reg 3.755000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 2.500050e-05 reg 4.004000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 2.500050e-05 reg 4.253000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 2.500050e-05 reg 4.502000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 2.500050e-05 reg 4.751000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 2.750045e-05 reg 2.000000e+04 train accuracy: 0.322306 val accurac
```

```
y: 0.332000
lr 2.750045e-05 reg 2.690000e+05 train accuracy: 0.100265 val accurac
y: 0.087000
lr 2.750045e-05 reg 5.180000e+05 train accuracy: 0.100265 val accurac
y: 0.087000
lr 2.750045e-05 reg 7.670000e+05 train accuracy: 0.100265 val accurac
y: 0.087000
lr 2.750045e-05 reg 1.016000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 2.750045e-05 reg 1.265000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 2.750045e-05 reg 1.514000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 2.750045e-05 reg 1.763000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 2.750045e-05 reg 2.012000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 2.750045e-05 reg 2.261000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 2.750045e-05 reg 2.510000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 2.750045e-05 reg 2.759000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 2.750045e-05 reg 3.008000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 2.750045e-05 reg 3.257000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 2.750045e-05 reg 3.506000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 2.750045e-05 reg 3.755000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 2.750045e-05 reg 4.004000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 2.750045e-05 reg 4.253000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 2.750045e-05 reg 4.502000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 2.750045e-05 reg 4.751000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 3.000040e-05 reg 2.000000e+04 train accuracy: 0.310265 val accurac
y: 0.332000
lr 3.000040e-05 reg 2.690000e+05 train accuracy: 0.100265 val accurac
y: 0.087000
lr 3.000040e-05 reg 5.180000e+05 train accuracy: 0.100265 val accurac
y: 0.087000
lr 3.000040e-05 reg 7.670000e+05 train accuracy: 0.100265 val accurac
y: 0.087000
lr 3.000040e-05 reg 1.016000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 3.000040e-05 reg 1.265000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 3.000040e-05 reg 1.514000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 3.000040e-05 reg 1.763000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 3.000040e-05 reg 2.012000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
```

```
lr 3.000040e-05 reg 2.261000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 3.000040e-05 reg 2.510000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 3.000040e-05 reg 2.759000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 3.000040e-05 reg 3.008000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 3.000040e-05 reg 3.257000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 3.000040e-05 reg 3.506000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 3.000040e-05 reg 3.755000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 3.000040e-05 reg 4.004000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 3.000040e-05 reg 4.253000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 3.000040e-05 reg 4.502000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 3.000040e-05 reg 4.751000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 3.250035e-05 reg 2.000000e+04 train accuracy: 0.323449 val accurac
y: 0.317000
lr 3.250035e-05 reg 2.690000e+05 train accuracy: 0.100265 val accurac
y: 0.087000
lr 3.250035e-05 reg 5.180000e+05 train accuracy: 0.100265 val accurac
y: 0.087000
lr 3.250035e-05 reg 7.670000e+05 train accuracy: 0.100265 val accurac
y: 0.087000
lr 3.250035e-05 reg 1.016000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 3.250035e-05 reg 1.265000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 3.250035e-05 reg 1.514000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 3.250035e-05 reg 1.763000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 3.250035e-05 reg 2.012000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 3.250035e-05 reg 2.261000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 3.250035e-05 reg 2.510000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 3.250035e-05 reg 2.759000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 3.250035e-05 reg 3.008000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 3.250035e-05 reg 3.257000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 3.250035e-05 reg 3.506000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 3.250035e-05 reg 3.755000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 3.250035e-05 reg 4.004000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 3.250035e-05 reg 4.253000e+06 train accuracy: 0.100265 val accurac
```

```
y: 0.087000
lr 3.250035e-05 reg 4.502000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 3.250035e-05 reg 4.751000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 3.500030e-05 reg 2.000000e+04 train accuracy: 0.293082 val accurac
y: 0.326000
lr 3.500030e-05 reg 2.690000e+05 train accuracy: 0.100265 val accurac
y: 0.087000
lr 3.500030e-05 reg 5.180000e+05 train accuracy: 0.100265 val accurac
y: 0.087000
lr 3.500030e-05 reg 7.670000e+05 train accuracy: 0.100265 val accurac
y: 0.087000
lr 3.500030e-05 reg 1.016000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 3.500030e-05 reg 1.265000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 3.500030e-05 reg 1.514000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 3.500030e-05 reg 1.763000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 3.500030e-05 reg 2.012000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 3.500030e-05 reg 2.261000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 3.500030e-05 reg 2.510000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 3.500030e-05 reg 2.759000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 3.500030e-05 reg 3.008000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 3.500030e-05 reg 3.257000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 3.500030e-05 reg 3.506000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 3.500030e-05 reg 3.755000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 3.500030e-05 reg 4.004000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 3.500030e-05 reg 4.253000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 3.500030e-05 reg 4.502000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 3.500030e-05 reg 4.751000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 3.750025e-05 reg 2.000000e+04 train accuracy: 0.288245 val accurac
y: 0.309000
lr 3.750025e-05 reg 2.690000e+05 train accuracy: 0.100265 val accurac
y: 0.087000
lr 3.750025e-05 reg 5.180000e+05 train accuracy: 0.100265 val accurac
y: 0.087000
lr 3.750025e-05 reg 7.670000e+05 train accuracy: 0.100265 val accurac
y: 0.087000
lr 3.750025e-05 reg 1.016000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 3.750025e-05 reg 1.265000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
```

```
lr 3.750025e-05 reg 1.514000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 3.750025e-05 reg 1.763000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 3.750025e-05 reg 2.012000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 3.750025e-05 reg 2.261000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 3.750025e-05 reg 2.510000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 3.750025e-05 reg 2.759000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 3.750025e-05 reg 3.008000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 3.750025e-05 reg 3.257000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 3.750025e-05 reg 3.506000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 3.750025e-05 reg 3.755000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 3.750025e-05 reg 4.004000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 3.750025e-05 reg 4.253000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 3.750025e-05 reg 4.502000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 3.750025e-05 reg 4.751000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 4.000020e-05 reg 2.000000e+04 train accuracy: 0.259469 val accurac
y: 0.258000
lr 4.000020e-05 reg 2.690000e+05 train accuracy: 0.100265 val accurac
y: 0.087000
lr 4.000020e-05 reg 5.180000e+05 train accuracy: 0.100265 val accurac
y: 0.087000
lr 4.000020e-05 reg 7.670000e+05 train accuracy: 0.100265 val accurac
y: 0.087000
lr 4.000020e-05 reg 1.016000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 4.000020e-05 reg 1.265000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 4.000020e-05 reg 1.514000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 4.000020e-05 reg 1.763000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 4.000020e-05 reg 2.012000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 4.000020e-05 reg 2.261000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 4.000020e-05 reg 2.510000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 4.000020e-05 reg 2.759000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 4.000020e-05 reg 3.008000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 4.000020e-05 reg 3.257000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 4.000020e-05 reg 3.506000e+06 train accuracy: 0.100265 val accurac
```

```
y: 0.087000
lr 4.000020e-05 reg 3.755000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 4.000020e-05 reg 4.004000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 4.000020e-05 reg 4.253000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 4.000020e-05 reg 4.502000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 4.000020e-05 reg 4.751000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 4.250015e-05 reg 2.000000e+04 train accuracy: 0.265082 val accurac
y: 0.277000
lr 4.250015e-05 reg 2.690000e+05 train accuracy: 0.100265 val accurac
y: 0.087000
lr 4.250015e-05 reg 5.180000e+05 train accuracy: 0.100265 val accurac
y: 0.087000
lr 4.250015e-05 reg 7.670000e+05 train accuracy: 0.100265 val accurac
y: 0.087000
lr 4.250015e-05 reg 1.016000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 4.250015e-05 reg 1.265000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 4.250015e-05 reg 1.514000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 4.250015e-05 reg 1.763000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 4.250015e-05 reg 2.012000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 4.250015e-05 reg 2.261000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 4.250015e-05 reg 2.510000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 4.250015e-05 reg 2.759000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 4.250015e-05 reg 3.008000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 4.250015e-05 reg 3.257000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 4.250015e-05 reg 3.506000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 4.250015e-05 reg 3.755000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 4.250015e-05 reg 4.004000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 4.250015e-05 reg 4.253000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 4.250015e-05 reg 4.502000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 4.250015e-05 reg 4.751000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 4.500010e-05 reg 2.000000e+04 train accuracy: 0.241735 val accurac
y: 0.260000
lr 4.500010e-05 reg 2.690000e+05 train accuracy: 0.100265 val accurac
y: 0.087000
lr 4.500010e-05 reg 5.180000e+05 train accuracy: 0.100265 val accurac
y: 0.087000
```

```
lr 4.500010e-05 reg 7.670000e+05 train accuracy: 0.100265 val accurac
y: 0.087000
lr 4.500010e-05 reg 1.016000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 4.500010e-05 reg 1.265000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 4.500010e-05 reg 1.514000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 4.500010e-05 reg 1.763000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 4.500010e-05 reg 2.012000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 4.500010e-05 reg 2.261000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 4.500010e-05 reg 2.510000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 4.500010e-05 reg 2.759000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 4.500010e-05 reg 3.008000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 4.500010e-05 reg 3.257000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 4.500010e-05 reg 3.506000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 4.500010e-05 reg 3.755000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 4.500010e-05 reg 4.004000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 4.500010e-05 reg 4.253000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 4.500010e-05 reg 4.502000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 4.500010e-05 reg 4.751000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 4.750005e-05 reg 2.000000e+04 train accuracy: 0.225286 val accurac
y: 0.252000
lr 4.750005e-05 reg 2.690000e+05 train accuracy: 0.100265 val accurac
y: 0.087000
lr 4.750005e-05 reg 5.180000e+05 train accuracy: 0.100265 val accurac
y: 0.087000
lr 4.750005e-05 reg 7.670000e+05 train accuracy: 0.100265 val accurac
y: 0.087000
lr 4.750005e-05 reg 1.016000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 4.750005e-05 reg 1.265000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 4.750005e-05 reg 1.514000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 4.750005e-05 reg 1.763000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 4.750005e-05 reg 2.012000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 4.750005e-05 reg 2.261000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 4.750005e-05 reg 2.510000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 4.750005e-05 reg 2.759000e+06 train accuracy: 0.100265 val accurac
```

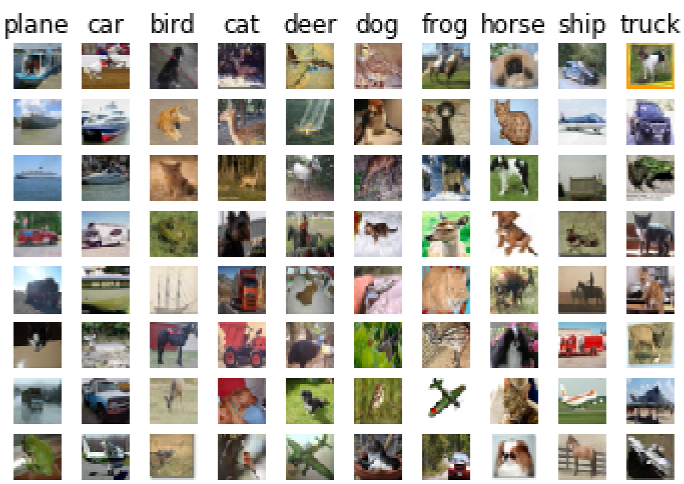
```
y: 0.087000
lr 4.750005e-05 reg 3.008000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 4.750005e-05 reg 3.257000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 4.750005e-05 reg 3.506000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 4.750005e-05 reg 3.755000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 4.750005e-05 reg 4.004000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 4.750005e-05 reg 4.253000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 4.750005e-05 reg 4.502000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
lr 4.750005e-05 reg 4.751000e+06 train accuracy: 0.100265 val accurac
y: 0.087000
best validation accuracy achieved during cross-validation: 0.423000
```

In [10]:

```
# 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)
```

```
0.418
```

In [11]:
```
# An important way to gain intuition about how an algorithm works is to
# visualize the mistakes that it makes. In this visualization, we show examples
# of images that are misclassified by our current system. The first column
# shows images that our system labeled as "plane" but whose true label is
# something other than "plane".

examples_per_class = 8
classes = ['plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']
for cls, cls_name in enumerate(classes):
    idxs = np.where((y_test != cls) & (y_test_pred == cls))[0]
    idxs = np.random.choice(idxs, examples_per_class, replace=False)
    for i, idx in enumerate(idxs):
        plt.subplot(examples_per_class, len(classes), i * len(classes) + cls + 1)
        plt.imshow(X_test[idx].astype('uint8'))
        plt.axis('off')
        if i == 0:
            plt.title(cls_name)
plt.show()
```



Inline question 1:

Describe the misclassification results that you see. Do they make sense?

They kind of make sense. For example, misclassificatoins in plane category were most probably becuase there is lot of sky and bluish tinge in those images. Similary, in car category, it put some ships and trucks, which look a lot like a car in some angles. So,yes. It does make sense.

Neural Network on image features

Earlier in this assigment we saw that training a two-layer neural network on raw pixels achieved better classification performance than linear classifiers on raw pixels. In this notebook we have seen that linear classifiers on image features outperform linear classifiers on raw pixels.

For completeness, we should also try training a neural network on image features. This approach should outperform all previous approaches: you should easily be able to achieve over 55% classification accuracy on the test set; our best model achieves about 60% classification accuracy.

In [12]:

```
# Preprocessing: Remove the bias dimension
# Make sure to run this cell only ONCE
print(X_train_feats.shape)
X_train_feats = X_train_feats[:, :-1]
X_val_feats = X_val_feats[:, :-1]
X_test_feats = X_test_feats[:, :-1]

print(X_train_feats.shape)
```

```
(49000, 155)
(49000, 154)
```

In [26]:
```
from cs682.classifiers.neural_net import TwoLayerNet

# input_dim = X_train_feats.shape[1]
# hidden_dim = 500
# num_classes = 10

# net = TwoLayerNet(input_dim, hidden_dim, num_classes)
# best_net = None

####################################################################
###########
# TODO: Train a two-layer neural network on image features. You may w
ant to    #
# cross-validate various parameters as in previous sections. Store yo
ur best   #
# model in the best_net variable.
#
####################################################################
###########
best_net = None # store the best model into this
input_size = X_train_feats.shape[1]
learning_rates = [1e-1, 9e-1]
regularization_strengths = [1e-2, 1e-1]
hiddenSizes = range(500,501)
num_classes = 10
results = {}
best_val = -1
best_lr = -1
best_reg = -1
best_hidden_size = -1
inner_iterations = 3000
batch_size = 200
learning_rate_decay=0.95

num_iters = 10
for it in range(num_iters):
    for jt in range(num_iters):
        for kt in range(len(hiddenSizes)):
            learning_rate = learning_rates[0] + it * ((learning_rates
[1] - learning_rates[0]) / num_iters)
            reg = regularization_strengths[0] + jt * ((regularization
_strengths[1] - regularization_strengths[0])/ num_iters)
            hidden_size = hiddenSizes[kt]

            net = TwoLayerNet(input_size, hidden_size, num_classes)
            # Train the network
            stats = net.train(X_train_feats, y_train, X_val_feats, y_
val,
                        learning_rate, learning_rate_decay,
                        reg, inner_iterations, batch_size,verbose=Fal
se)

            y_train_pred = net.predict(X_train_feats)
            y_val_pred = net.predict(X_val_feats)
            training_accuracy = np.mean(y_train == y_train_pred)
            validation_accuracy = np.mean(y_val == y_val_pred)
```

```
            results[(learning_rate, reg, hidden_size)] = (training_ac
curacy, validation_accuracy)
            print('lr %e reg %e hidden %f train accuracy: %f val accu
racy: %f' % (
                        learning_rate, reg, hidden_size, training_acc
uracy, validation_accuracy))
            if validation_accuracy > best_val:
                best_val = validation_accuracy
                best_net = net
                best_lr = learning_rate
                best_reg = reg
                best_hidden_size = hidden_size

print('best validation accuracy achieved during cross-validation: lr
%e reg %e hidden %f val accuracy: %f, ' % (learning_rate, reg, hidden
_size, best_val))


#####################################################################
###########
#                              END OF YOUR CODE
#
#####################################################################
###########
```

```
lr 1.000000e-01 reg 1.000000e-02 hidden 500.000000 train accuracy: 0.
523612 val accuracy: 0.517000
lr 1.000000e-01 reg 1.900000e-02 hidden 500.000000 train accuracy: 0.
504714 val accuracy: 0.503000
lr 1.000000e-01 reg 2.800000e-02 hidden 500.000000 train accuracy: 0.
491878 val accuracy: 0.489000
lr 1.000000e-01 reg 3.700000e-02 hidden 500.000000 train accuracy: 0.
468082 val accuracy: 0.466000
lr 1.000000e-01 reg 4.600000e-02 hidden 500.000000 train accuracy: 0.
451755 val accuracy: 0.454000
lr 1.000000e-01 reg 5.500000e-02 hidden 500.000000 train accuracy: 0.
435306 val accuracy: 0.423000
lr 1.000000e-01 reg 6.400000e-02 hidden 500.000000 train accuracy: 0.
416347 val accuracy: 0.410000
lr 1.000000e-01 reg 7.300000e-02 hidden 500.000000 train accuracy: 0.
389245 val accuracy: 0.390000
lr 1.000000e-01 reg 8.200000e-02 hidden 500.000000 train accuracy: 0.
364878 val accuracy: 0.370000
lr 1.000000e-01 reg 9.100000e-02 hidden 500.000000 train accuracy: 0.
346286 val accuracy: 0.346000
lr 1.800000e-01 reg 1.000000e-02 hidden 500.000000 train accuracy: 0.
527898 val accuracy: 0.507000
lr 1.800000e-01 reg 1.900000e-02 hidden 500.000000 train accuracy: 0.
510082 val accuracy: 0.510000
lr 1.800000e-01 reg 2.800000e-02 hidden 500.000000 train accuracy: 0.
490653 val accuracy: 0.497000
lr 1.800000e-01 reg 3.700000e-02 hidden 500.000000 train accuracy: 0.
466449 val accuracy: 0.456000
lr 1.800000e-01 reg 4.600000e-02 hidden 500.000000 train accuracy: 0.
449571 val accuracy: 0.451000
lr 1.800000e-01 reg 5.500000e-02 hidden 500.000000 train accuracy: 0.
433388 val accuracy: 0.435000
lr 1.800000e-01 reg 6.400000e-02 hidden 500.000000 train accuracy: 0.
413633 val accuracy: 0.413000
lr 1.800000e-01 reg 7.300000e-02 hidden 500.000000 train accuracy: 0.
402490 val accuracy: 0.396000
lr 1.800000e-01 reg 8.200000e-02 hidden 500.000000 train accuracy: 0.
371347 val accuracy: 0.355000
lr 1.800000e-01 reg 9.100000e-02 hidden 500.000000 train accuracy: 0.
365612 val accuracy: 0.344000
lr 2.600000e-01 reg 1.000000e-02 hidden 500.000000 train accuracy: 0.
532612 val accuracy: 0.522000
lr 2.600000e-01 reg 1.900000e-02 hidden 500.000000 train accuracy: 0.
497429 val accuracy: 0.497000
lr 2.600000e-01 reg 2.800000e-02 hidden 500.000000 train accuracy: 0.
490204 val accuracy: 0.474000
lr 2.600000e-01 reg 3.700000e-02 hidden 500.000000 train accuracy: 0.
460694 val accuracy: 0.456000
lr 2.600000e-01 reg 4.600000e-02 hidden 500.000000 train accuracy: 0.
446694 val accuracy: 0.458000
lr 2.600000e-01 reg 5.500000e-02 hidden 500.000000 train accuracy: 0.
422469 val accuracy: 0.419000
lr 2.600000e-01 reg 6.400000e-02 hidden 500.000000 train accuracy: 0.
409878 val accuracy: 0.408000
lr 2.600000e-01 reg 7.300000e-02 hidden 500.000000 train accuracy: 0.
405327 val accuracy: 0.424000
lr 2.600000e-01 reg 8.200000e-02 hidden 500.000000 train accuracy: 0.
```

```
369816 val accuracy: 0.353000
lr 2.600000e-01 reg 9.100000e-02 hidden 500.000000 train accuracy: 0.
361673 val accuracy: 0.361000
lr 3.400000e-01 reg 1.000000e-02 hidden 500.000000 train accuracy: 0.
528469 val accuracy: 0.522000
lr 3.400000e-01 reg 1.900000e-02 hidden 500.000000 train accuracy: 0.
503367 val accuracy: 0.484000
lr 3.400000e-01 reg 2.800000e-02 hidden 500.000000 train accuracy: 0.
491898 val accuracy: 0.493000
lr 3.400000e-01 reg 3.700000e-02 hidden 500.000000 train accuracy: 0.
455163 val accuracy: 0.421000
lr 3.400000e-01 reg 4.600000e-02 hidden 500.000000 train accuracy: 0.
447102 val accuracy: 0.450000
lr 3.400000e-01 reg 5.500000e-02 hidden 500.000000 train accuracy: 0.
409980 val accuracy: 0.407000
lr 3.400000e-01 reg 6.400000e-02 hidden 500.000000 train accuracy: 0.
413041 val accuracy: 0.391000
lr 3.400000e-01 reg 7.300000e-02 hidden 500.000000 train accuracy: 0.
392122 val accuracy: 0.372000
lr 3.400000e-01 reg 8.200000e-02 hidden 500.000000 train accuracy: 0.
367980 val accuracy: 0.354000
lr 3.400000e-01 reg 9.100000e-02 hidden 500.000000 train accuracy: 0.
366531 val accuracy: 0.342000
lr 4.200000e-01 reg 1.000000e-02 hidden 500.000000 train accuracy: 0.
528367 val accuracy: 0.521000
lr 4.200000e-01 reg 1.900000e-02 hidden 500.000000 train accuracy: 0.
502755 val accuracy: 0.501000
lr 4.200000e-01 reg 2.800000e-02 hidden 500.000000 train accuracy: 0.
484735 val accuracy: 0.475000
lr 4.200000e-01 reg 3.700000e-02 hidden 500.000000 train accuracy: 0.
457939 val accuracy: 0.451000
lr 4.200000e-01 reg 4.600000e-02 hidden 500.000000 train accuracy: 0.
432306 val accuracy: 0.429000
lr 4.200000e-01 reg 5.500000e-02 hidden 500.000000 train accuracy: 0.
411878 val accuracy: 0.405000
lr 4.200000e-01 reg 6.400000e-02 hidden 500.000000 train accuracy: 0.
398673 val accuracy: 0.377000
lr 4.200000e-01 reg 7.300000e-02 hidden 500.000000 train accuracy: 0.
382571 val accuracy: 0.370000
lr 4.200000e-01 reg 8.200000e-02 hidden 500.000000 train accuracy: 0.
362898 val accuracy: 0.347000
lr 4.200000e-01 reg 9.100000e-02 hidden 500.000000 train accuracy: 0.
273204 val accuracy: 0.247000
lr 5.000000e-01 reg 1.000000e-02 hidden 500.000000 train accuracy: 0.
526612 val accuracy: 0.521000
lr 5.000000e-01 reg 1.900000e-02 hidden 500.000000 train accuracy: 0.
495714 val accuracy: 0.491000
lr 5.000000e-01 reg 2.800000e-02 hidden 500.000000 train accuracy: 0.
481082 val accuracy: 0.484000
lr 5.000000e-01 reg 3.700000e-02 hidden 500.000000 train accuracy: 0.
447898 val accuracy: 0.429000
lr 5.000000e-01 reg 4.600000e-02 hidden 500.000000 train accuracy: 0.
426143 val accuracy: 0.446000
lr 5.000000e-01 reg 5.500000e-02 hidden 500.000000 train accuracy: 0.
413286 val accuracy: 0.415000
lr 5.000000e-01 reg 6.400000e-02 hidden 500.000000 train accuracy: 0.
391000 val accuracy: 0.384000
```

```
lr 5.000000e-01 reg 7.300000e-02 hidden 500.000000 train accuracy: 0.
400184 val accuracy: 0.398000
lr 5.000000e-01 reg 8.200000e-02 hidden 500.000000 train accuracy: 0.
374673 val accuracy: 0.369000
lr 5.000000e-01 reg 9.100000e-02 hidden 500.000000 train accuracy: 0.
339122 val accuracy: 0.320000
lr 5.800000e-01 reg 1.000000e-02 hidden 500.000000 train accuracy: 0.
526265 val accuracy: 0.534000
lr 5.800000e-01 reg 1.900000e-02 hidden 500.000000 train accuracy: 0.
491653 val accuracy: 0.488000
lr 5.800000e-01 reg 2.800000e-02 hidden 500.000000 train accuracy: 0.
465327 val accuracy: 0.464000
lr 5.800000e-01 reg 3.700000e-02 hidden 500.000000 train accuracy: 0.
458714 val accuracy: 0.457000
lr 5.800000e-01 reg 4.600000e-02 hidden 500.000000 train accuracy: 0.
419102 val accuracy: 0.409000
lr 5.800000e-01 reg 5.500000e-02 hidden 500.000000 train accuracy: 0.
400122 val accuracy: 0.400000
lr 5.800000e-01 reg 6.400000e-02 hidden 500.000000 train accuracy: 0.
394898 val accuracy: 0.371000
lr 5.800000e-01 reg 7.300000e-02 hidden 500.000000 train accuracy: 0.
377776 val accuracy: 0.369000
lr 5.800000e-01 reg 8.200000e-02 hidden 500.000000 train accuracy: 0.
347612 val accuracy: 0.349000
lr 5.800000e-01 reg 9.100000e-02 hidden 500.000000 train accuracy: 0.
347000 val accuracy: 0.339000
lr 6.600000e-01 reg 1.000000e-02 hidden 500.000000 train accuracy: 0.
518490 val accuracy: 0.504000
lr 6.600000e-01 reg 1.900000e-02 hidden 500.000000 train accuracy: 0.
487143 val accuracy: 0.507000
lr 6.600000e-01 reg 2.800000e-02 hidden 500.000000 train accuracy: 0.
467694 val accuracy: 0.462000
lr 6.600000e-01 reg 3.700000e-02 hidden 500.000000 train accuracy: 0.
439918 val accuracy: 0.452000
lr 6.600000e-01 reg 4.600000e-02 hidden 500.000000 train accuracy: 0.
431939 val accuracy: 0.413000
lr 6.600000e-01 reg 5.500000e-02 hidden 500.000000 train accuracy: 0.
404469 val accuracy: 0.413000
lr 6.600000e-01 reg 6.400000e-02 hidden 500.000000 train accuracy: 0.
403755 val accuracy: 0.413000
lr 6.600000e-01 reg 7.300000e-02 hidden 500.000000 train accuracy: 0.
312184 val accuracy: 0.315000
lr 6.600000e-01 reg 8.200000e-02 hidden 500.000000 train accuracy: 0.
359612 val accuracy: 0.341000
lr 6.600000e-01 reg 9.100000e-02 hidden 500.000000 train accuracy: 0.
327694 val accuracy: 0.296000
lr 7.400000e-01 reg 1.000000e-02 hidden 500.000000 train accuracy: 0.
513102 val accuracy: 0.502000
lr 7.400000e-01 reg 1.900000e-02 hidden 500.000000 train accuracy: 0.
473449 val accuracy: 0.475000
lr 7.400000e-01 reg 2.800000e-02 hidden 500.000000 train accuracy: 0.
481469 val accuracy: 0.472000
lr 7.400000e-01 reg 3.700000e-02 hidden 500.000000 train accuracy: 0.
443571 val accuracy: 0.409000
lr 7.400000e-01 reg 4.600000e-02 hidden 500.000000 train accuracy: 0.
426714 val accuracy: 0.436000
lr 7.400000e-01 reg 5.500000e-02 hidden 500.000000 train accuracy: 0.
```

```
383694 val accuracy: 0.386000
lr 7.400000e-01 reg 6.400000e-02 hidden 500.000000 train accuracy: 0.
386776 val accuracy: 0.390000
lr 7.400000e-01 reg 7.300000e-02 hidden 500.000000 train accuracy: 0.
368551 val accuracy: 0.353000
lr 7.400000e-01 reg 8.200000e-02 hidden 500.000000 train accuracy: 0.
325898 val accuracy: 0.307000
lr 7.400000e-01 reg 9.100000e-02 hidden 500.000000 train accuracy: 0.
305388 val accuracy: 0.307000
lr 8.200000e-01 reg 1.000000e-02 hidden 500.000000 train accuracy: 0.
518673 val accuracy: 0.500000
lr 8.200000e-01 reg 1.900000e-02 hidden 500.000000 train accuracy: 0.
473633 val accuracy: 0.489000
lr 8.200000e-01 reg 2.800000e-02 hidden 500.000000 train accuracy: 0.
459327 val accuracy: 0.448000
lr 8.200000e-01 reg 3.700000e-02 hidden 500.000000 train accuracy: 0.
446082 val accuracy: 0.446000
lr 8.200000e-01 reg 4.600000e-02 hidden 500.000000 train accuracy: 0.
436224 val accuracy: 0.436000
lr 8.200000e-01 reg 5.500000e-02 hidden 500.000000 train accuracy: 0.
403837 val accuracy: 0.391000
lr 8.200000e-01 reg 6.400000e-02 hidden 500.000000 train accuracy: 0.
387408 val accuracy: 0.368000
lr 8.200000e-01 reg 7.300000e-02 hidden 500.000000 train accuracy: 0.
361102 val accuracy: 0.374000
lr 8.200000e-01 reg 8.200000e-02 hidden 500.000000 train accuracy: 0.
359122 val accuracy: 0.343000
lr 8.200000e-01 reg 9.100000e-02 hidden 500.000000 train accuracy: 0.
331408 val accuracy: 0.314000
best validation accuracy achieved during cross-validation: lr 8.20000
0e-01 reg 9.100000e-02 hidden 500.000000 val accuracy: 0.534000,
```

In [27]:

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

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

```
0.514
```

In [ ]: