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 (http://vision.stanford.edu/teaching/cs231n/assignments.html)](http://vision.stanford.edu/teaching/cs231n/assignments.html) 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 [26]:

```
import random
import numpy as np
from cs231n.data_utils import load_CIFAR10
import matplotlib.pyplot as plt

from __future__ import print_function

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

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

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

Load data

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

In [27]:

```
from cs231n.features import color_histogram_hsv, hog_feature

def get_CIFAR10_data(num_training=49000, num_validation=1000, num_test=1000):
    # Load the raw CIFAR-10 data
    cifar10_dir = 'cs231n/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()
```

```
Clear previously loaded 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 [28]:

```
from cs231n.features import *

# 10, 11,
num_color_bins = 16 # 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.shape[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 [13]:

```
# Use the validation set to tune the learning rate and regularization strength

from cs231n.classifiers.linear_classifier import LinearSVM

learning_rates = [5e-8, 1e-7, 2e-7, 3e-7, 4e-7, 5e-7, 1e-6, 5e-6]
regularization_strengths = [1e3, 5e3, 1e4, 2e4, 3e4, 4e4, 5e4, 1e5]

results = {}
best_val = -1   # The highest validation accuracy that we have seen so far.
best_svm = None # The LinearSVM object that achieved the highest validation rate
.
best_params = 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 SVM; 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 careful    #
# you should be able to get accuracy of near 0.44 on the validation set.       #
################################################################################
for l in learning_rates:
    for r in regularization_strengths:
        svm = LinearSVM()
        curr_loss = svm.train(X_train_feats, y_train, learning_rate=l, reg=r,
                      num_iters=1500, verbose=True)
        y_train_pred = svm.predict(X_train_feats)
        y_train_acc = np.mean(y_train == y_train_pred)
        y_val_pred = svm.predict(X_val_feats)
        y_val_acc = np.mean(y_val == y_val_pred)
        results[(l, r)] = (y_train_acc, y_val_acc)
        if y_val_acc > best_val:
            best_svm = svm
            best_val = y_val_acc
            best_params = (l, r)

################################################################################
#                              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
)
print best_params
```

iteration 0 / 1500: loss 10.596329

```
iteration 0 / 1500: loss 10.590329
iteration 100 / 1500: loss 10.578521
iteration 200 / 1500: loss 10.546396
iteration 300 / 1500: loss 10.499155
iteration 400 / 1500: loss 10.469077
iteration 500 / 1500: loss 10.447657
iteration 600 / 1500: loss 10.414133
iteration 700 / 1500: loss 10.401073
iteration 800 / 1500: loss 10.364895
iteration 900 / 1500: loss 10.327907
iteration 1000 / 1500: loss 10.310513
iteration 1100 / 1500: loss 10.287885
iteration 1200 / 1500: loss 10.259661
iteration 1300 / 1500: loss 10.233150
iteration 1400 / 1500: loss 10.191958
iteration 0 / 1500: loss 16.635261
iteration 100 / 1500: loss 15.920372
iteration 200 / 1500: loss 15.261402
iteration 300 / 1500: loss 14.666318
iteration 400 / 1500: loss 14.126514
iteration 500 / 1500: loss 13.648507
iteration 600 / 1500: loss 13.197804
iteration 700 / 1500: loss 12.798200
iteration 800 / 1500: loss 12.440185
iteration 900 / 1500: loss 12.117257
iteration 1000 / 1500: loss 11.824610
iteration 1100 / 1500: loss 11.544930
iteration 1200 / 1500: loss 11.300652
iteration 1300 / 1500: loss 11.081145
iteration 1400 / 1500: loss 10.881673
iteration 0 / 1500: loss 24.597960
iteration 100 / 1500: loss 21.769524
iteration 200 / 1500: loss 19.453502
iteration 300 / 1500: loss 17.565047
iteration 400 / 1500: loss 16.009198
iteration 500 / 1500: loss 14.734748
iteration 600 / 1500: loss 13.695182
iteration 700 / 1500: loss 12.845738
iteration 800 / 1500: loss 12.145728
iteration 900 / 1500: loss 11.574274
iteration 1000 / 1500: loss 11.106844
iteration 1100 / 1500: loss 10.726786
iteration 1200 / 1500: loss 10.413845
iteration 1300 / 1500: loss 10.156120
iteration 1400 / 1500: loss 9.942285
iteration 0 / 1500: loss 41.215168
iteration 100 / 1500: loss 30.586929
iteration 200 / 1500: loss 23.459343
iteration 300 / 1500: loss 18.687359
iteration 400 / 1500: loss 15.492110
iteration 500 / 1500: loss 13.353427
iteration 600 / 1500: loss 11.912268
iteration 700 / 1500: loss 10.955144
iteration 800 / 1500: loss 10.307624
```

```
iteration 900 / 1500: loss 9.876250
iteration 1000 / 1500: loss 9.587906
iteration 1100 / 1500: loss 9.392915
iteration 1200 / 1500: loss 9.264058
iteration 1300 / 1500: loss 9.175502
iteration 1400 / 1500: loss 9.116656
iteration 0 / 1500: loss 56.992314
iteration 100 / 1500: loss 35.318036
iteration 200 / 1500: loss 23.428484
iteration 300 / 1500: loss 16.912829
iteration 400 / 1500: loss 13.338948
iteration 500 / 1500: loss 11.377607
iteration 600 / 1500: loss 10.304441
iteration 700 / 1500: loss 9.713867
iteration 800 / 1500: loss 9.392074
iteration 900 / 1500: loss 9.214761
iteration 1000 / 1500: loss 9.117002
iteration 1100 / 1500: loss 9.064144
iteration 1200 / 1500: loss 9.034676
iteration 1300 / 1500: loss 9.018974
iteration 1400 / 1500: loss 9.009795
iteration 0 / 1500: loss 71.806144
iteration 100 / 1500: loss 37.177618
iteration 200 / 1500: loss 21.640609
iteration 300 / 1500: loss 14.671909
iteration 400 / 1500: loss 11.544970
iteration 500 / 1500: loss 10.142297
iteration 600 / 1500: loss 9.512133
iteration 700 / 1500: loss 9.229340
iteration 800 / 1500: loss 9.102448
iteration 900 / 1500: loss 9.045769
iteration 1000 / 1500: loss 9.020374
iteration 1100 / 1500: loss 9.009014
iteration 1200 / 1500: loss 9.003781
iteration 1300 / 1500: loss 9.001471
iteration 1400 / 1500: loss 9.000446
iteration 0 / 1500: loss 88.089470
iteration 100 / 1500: loss 38.027785
iteration 200 / 1500: loss 19.648601
iteration 300 / 1500: loss 12.908922
iteration 400 / 1500: loss 10.437089
iteration 500 / 1500: loss 9.525816
iteration 600 / 1500: loss 9.192766
iteration 700 / 1500: loss 9.070484
iteration 800 / 1500: loss 9.025501
iteration 900 / 1500: loss 9.009187
iteration 1000 / 1500: loss 9.003053
iteration 1100 / 1500: loss 9.000979
iteration 1200 / 1500: loss 9.000162
iteration 1300 / 1500: loss 8.999800
iteration 1400 / 1500: loss 8.999701
iteration 0 / 1500: loss 171.061714
iteration 100 / 1500: loss 30.716381
```

```
iteration 200 / 1500: loss 11.909215
iteration 300 / 1500: loss 9.389579
iteration 400 / 1500: loss 9.052185
iteration 500 / 1500: loss 9.006856
iteration 600 / 1500: loss 9.000763
iteration 700 / 1500: loss 8.999907
iteration 800 / 1500: loss 8.999812
iteration 900 / 1500: loss 8.999824
iteration 1000 / 1500: loss 8.999816
iteration 1100 / 1500: loss 8.999858
iteration 1200 / 1500: loss 8.999816
iteration 1300 / 1500: loss 8.999841
iteration 1400 / 1500: loss 8.999859
iteration 0 / 1500: loss 10.649769
iteration 100 / 1500: loss 10.601363
iteration 200 / 1500: loss 10.510027
iteration 300 / 1500: loss 10.454438
iteration 400 / 1500: loss 10.401835
iteration 500 / 1500: loss 10.344445
iteration 600 / 1500: loss 10.300307
iteration 700 / 1500: loss 10.260208
iteration 800 / 1500: loss 10.193068
iteration 900 / 1500: loss 10.161515
iteration 1000 / 1500: loss 10.106179
iteration 1100 / 1500: loss 10.050118
iteration 1200 / 1500: loss 10.007931
iteration 1300 / 1500: loss 9.982270
iteration 1400 / 1500: loss 9.938138
iteration 0 / 1500: loss 17.387181
iteration 100 / 1500: loss 15.866039
iteration 200 / 1500: loss 14.617982
iteration 300 / 1500: loss 13.600652
iteration 400 / 1500: loss 12.771689
iteration 500 / 1500: loss 12.079819
iteration 600 / 1500: loss 11.524240
iteration 700 / 1500: loss 11.064045
iteration 800 / 1500: loss 10.686178
iteration 900 / 1500: loss 10.379494
iteration 1000 / 1500: loss 10.134963
iteration 1100 / 1500: loss 9.930588
iteration 1200 / 1500: loss 9.757500
iteration 1300 / 1500: loss 9.618199
iteration 1400 / 1500: loss 9.504882
iteration 0 / 1500: loss 24.795060
iteration 100 / 1500: loss 19.577447
iteration 200 / 1500: loss 16.091228
iteration 300 / 1500: loss 13.756382
iteration 400 / 1500: loss 12.181977
iteration 500 / 1500: loss 11.134599
iteration 600 / 1500: loss 10.432101
iteration 700 / 1500: loss 9.959688
iteration 800 / 1500: loss 9.640130
iteration 900 / 1500: loss 9.429855
```

```
iteration 1000 / 1500: loss 9.287308
iteration 1100 / 1500: loss 9.190578
iteration 1200 / 1500: loss 9.125790
iteration 1300 / 1500: loss 9.084592
iteration 1400 / 1500: loss 9.056440
iteration 0 / 1500: loss 41.984678
iteration 100 / 1500: loss 23.792966
iteration 200 / 1500: loss 15.632379
iteration 300 / 1500: loss 11.978102
iteration 400 / 1500: loss 10.336052
iteration 500 / 1500: loss 9.596862
iteration 600 / 1500: loss 9.268282
iteration 700 / 1500: loss 9.120058
iteration 800 / 1500: loss 9.053599
iteration 900 / 1500: loss 9.022866
iteration 1000 / 1500: loss 9.009820
iteration 1100 / 1500: loss 9.003661
iteration 1200 / 1500: loss 9.001228
iteration 1300 / 1500: loss 9.000169
iteration 1400 / 1500: loss 8.999427
iteration 0 / 1500: loss 54.622317
iteration 100 / 1500: loss 22.694929
iteration 200 / 1500: loss 13.106683
iteration 300 / 1500: loss 10.235222
iteration 400 / 1500: loss 9.369835
iteration 500 / 1500: loss 9.110756
iteration 600 / 1500: loss 9.032846
iteration 700 / 1500: loss 9.009534
iteration 800 / 1500: loss 9.002205
iteration 900 / 1500: loss 9.000426
iteration 1000 / 1500: loss 8.999723
iteration 1100 / 1500: loss 8.999557
iteration 1200 / 1500: loss 8.999541
iteration 1300 / 1500: loss 8.999424
iteration 1400 / 1500: loss 8.999423
iteration 0 / 1500: loss 73.772330
iteration 100 / 1500: loss 21.988922
iteration 200 / 1500: loss 11.608119
iteration 300 / 1500: loss 9.521271
iteration 400 / 1500: loss 9.104405
iteration 500 / 1500: loss 9.020349
iteration 600 / 1500: loss 9.003797
iteration 700 / 1500: loss 9.000310
iteration 800 / 1500: loss 8.999705
iteration 900 / 1500: loss 8.999640
iteration 1000 / 1500: loss 8.999554
iteration 1100 / 1500: loss 8.999570
iteration 1200 / 1500: loss 8.999551
iteration 1300 / 1500: loss 8.999696
iteration 1400 / 1500: loss 8.999587
iteration 0 / 1500: loss 87.630900
iteration 100 / 1500: loss 19.528651
iteration 200 / 1500: loss 10.410475
```

```
iteration 300 / 1500: loss 9.188930
iteration 400 / 1500: loss 9.024899
iteration 500 / 1500: loss 9.003082
iteration 600 / 1500: loss 9.000108
iteration 700 / 1500: loss 8.999695
iteration 800 / 1500: loss 8.999671
iteration 900 / 1500: loss 8.999654
iteration 1000 / 1500: loss 8.999692
iteration 1100 / 1500: loss 8.999623
iteration 1200 / 1500: loss 8.999664
iteration 1300 / 1500: loss 8.999582
iteration 1400 / 1500: loss 8.999663
iteration 0 / 1500: loss 169.096718
iteration 100 / 1500: loss 11.814801
iteration 200 / 1500: loss 9.049346
iteration 300 / 1500: loss 9.000670
iteration 400 / 1500: loss 8.999870
iteration 500 / 1500: loss 8.999839
iteration 600 / 1500: loss 8.999842
iteration 700 / 1500: loss 8.999862
iteration 800 / 1500: loss 8.999791
iteration 900 / 1500: loss 8.999797
iteration 1000 / 1500: loss 8.999793
iteration 1100 / 1500: loss 8.999786
iteration 1200 / 1500: loss 8.999813
iteration 1300 / 1500: loss 8.999809
iteration 1400 / 1500: loss 8.999826
iteration 0 / 1500: loss 10.604011
iteration 100 / 1500: loss 10.472676
iteration 200 / 1500: loss 10.364896
iteration 300 / 1500: loss 10.261710
iteration 400 / 1500: loss 10.167433
iteration 500 / 1500: loss 10.074861
iteration 600 / 1500: loss 9.979947
iteration 700 / 1500: loss 9.914838
iteration 800 / 1500: loss 9.828529
iteration 900 / 1500: loss 9.781575
iteration 1000 / 1500: loss 9.710182
iteration 1100 / 1500: loss 9.670637
iteration 1200 / 1500: loss 9.611616
iteration 1300 / 1500: loss 9.563630
iteration 1400 / 1500: loss 9.509502
iteration 0 / 1500: loss 16.966789
iteration 100 / 1500: loss 14.350894
iteration 200 / 1500: loss 12.579941
iteration 300 / 1500: loss 11.394810
iteration 400 / 1500: loss 10.613195
iteration 500 / 1500: loss 10.076459
iteration 600 / 1500: loss 9.720416
iteration 700 / 1500: loss 9.480665
iteration 800 / 1500: loss 9.319702
iteration 900 / 1500: loss 9.213474
iteration 1000 / 1500: loss 9.142741
```

```
iteration 1100 / 1500: loss 9.094373
iteration 1200 / 1500: loss 9.061986
iteration 1300 / 1500: loss 9.040417
iteration 1400 / 1500: loss 9.025286
iteration 0 / 1500: loss 25.081834
iteration 100 / 1500: loss 16.212045
iteration 200 / 1500: loss 12.230897
iteration 300 / 1500: loss 10.446150
iteration 400 / 1500: loss 9.647403
iteration 500 / 1500: loss 9.290825
iteration 600 / 1500: loss 9.128026
iteration 700 / 1500: loss 9.057532
iteration 800 / 1500: loss 9.024519
iteration 900 / 1500: loss 9.010335
iteration 1000 / 1500: loss 9.003811
iteration 1100 / 1500: loss 9.000578
iteration 1200 / 1500: loss 8.999315
iteration 1300 / 1500: loss 8.998705
iteration 1400 / 1500: loss 8.998336
iteration 0 / 1500: loss 41.144295
iteration 100 / 1500: loss 15.447825
iteration 200 / 1500: loss 10.292492
iteration 300 / 1500: loss 9.259019
iteration 400 / 1500: loss 9.051028
iteration 500 / 1500: loss 9.009489
iteration 600 / 1500: loss 9.001051
iteration 700 / 1500: loss 8.999504
iteration 800 / 1500: loss 8.999237
iteration 900 / 1500: loss 8.999150
iteration 1000 / 1500: loss 8.999052
iteration 1100 / 1500: loss 8.999022
iteration 1200 / 1500: loss 8.999202
iteration 1300 / 1500: loss 8.999341
iteration 1400 / 1500: loss 8.999178
iteration 0 / 1500: loss 58.600333
iteration 100 / 1500: loss 13.433920
iteration 200 / 1500: loss 9.395946
iteration 300 / 1500: loss 9.034582
iteration 400 / 1500: loss 9.002482
iteration 500 / 1500: loss 8.999674
iteration 600 / 1500: loss 8.999379
iteration 700 / 1500: loss 8.999499
iteration 800 / 1500: loss 8.999400
iteration 900 / 1500: loss 8.999474
iteration 1000 / 1500: loss 8.999220
iteration 1100 / 1500: loss 8.999395
iteration 1200 / 1500: loss 8.999348
iteration 1300 / 1500: loss 8.999499
iteration 1400 / 1500: loss 8.999431
iteration 0 / 1500: loss 75.681033
iteration 100 / 1500: loss 11.651563
iteration 200 / 1500: loss 9.104614
iteration 300 / 1500: loss 9.003786
```

```
iteration 400 / 1500: loss 8.999635
iteration 500 / 1500: loss 8.999590
iteration 600 / 1500: loss 8.999578
iteration 700 / 1500: loss 8.999654
iteration 800 / 1500: loss 8.999546
iteration 900 / 1500: loss 8.999539
iteration 1000 / 1500: loss 8.999626
iteration 1100 / 1500: loss 8.999588
iteration 1200 / 1500: loss 8.999535
iteration 1300 / 1500: loss 8.999426
iteration 1400 / 1500: loss 8.999521
iteration 0 / 1500: loss 90.584749
iteration 100 / 1500: loss 10.432575
iteration 200 / 1500: loss 9.025119
iteration 300 / 1500: loss 9.000137
iteration 400 / 1500: loss 8.999540
iteration 500 / 1500: loss 8.999557
iteration 600 / 1500: loss 8.999643
iteration 700 / 1500: loss 8.999586
iteration 800 / 1500: loss 8.999671
iteration 900 / 1500: loss 8.999660
iteration 1000 / 1500: loss 8.999664
iteration 1100 / 1500: loss 8.999622
iteration 1200 / 1500: loss 8.999616
iteration 1300 / 1500: loss 8.999588
iteration 1400 / 1500: loss 8.999640
iteration 0 / 1500: loss 173.694308
iteration 100 / 1500: loss 9.046457
iteration 200 / 1500: loss 8.999853
iteration 300 / 1500: loss 8.999820
iteration 400 / 1500: loss 8.999819
iteration 500 / 1500: loss 8.999787
iteration 600 / 1500: loss 8.999841
iteration 700 / 1500: loss 8.999827
iteration 800 / 1500: loss 8.999826
iteration 900 / 1500: loss 8.999828
iteration 1000 / 1500: loss 8.999815
iteration 1100 / 1500: loss 8.999838
iteration 1200 / 1500: loss 8.999837
iteration 1300 / 1500: loss 8.999824
iteration 1400 / 1500: loss 8.999834
iteration 0 / 1500: loss 10.597976
iteration 100 / 1500: loss 10.422787
iteration 200 / 1500: loss 10.261017
iteration 300 / 1500: loss 10.116479
iteration 400 / 1500: loss 9.975429
iteration 500 / 1500: loss 9.873461
iteration 600 / 1500: loss 9.773045
iteration 700 / 1500: loss 9.676795
iteration 800 / 1500: loss 9.597652
iteration 900 / 1500: loss 9.521525
iteration 1000 / 1500: loss 9.463032
iteration 1100 / 1500: loss 9.420537
```

```
iteration 1200 / 1500: loss 9.366744
iteration 1300 / 1500: loss 9.323150
iteration 1400 / 1500: loss 9.282847
iteration 0 / 1500: loss 17.233463
iteration 100 / 1500: loss 13.505898
iteration 200 / 1500: loss 11.479238
iteration 300 / 1500: loss 10.357619
iteration 400 / 1500: loss 9.741784
iteration 500 / 1500: loss 9.404683
iteration 600 / 1500: loss 9.222597
iteration 700 / 1500: loss 9.120459
iteration 800 / 1500: loss 9.062282
iteration 900 / 1500: loss 9.032998
iteration 1000 / 1500: loss 9.016069
iteration 1100 / 1500: loss 9.007595
iteration 1200 / 1500: loss 9.002323
iteration 1300 / 1500: loss 8.999683
iteration 1400 / 1500: loss 8.998030
iteration 0 / 1500: loss 25.399315
iteration 100 / 1500: loss 13.916744
iteration 200 / 1500: loss 10.472596
iteration 300 / 1500: loss 9.441882
iteration 400 / 1500: loss 9.129961
iteration 500 / 1500: loss 9.038050
iteration 600 / 1500: loss 9.010227
iteration 700 / 1500: loss 9.001496
iteration 800 / 1500: loss 8.999173
iteration 900 / 1500: loss 8.998699
iteration 1000 / 1500: loss 8.997935
iteration 1100 / 1500: loss 8.998308
iteration 1200 / 1500: loss 8.998244
iteration 1300 / 1500: loss 8.998004
iteration 1400 / 1500: loss 8.998457
iteration 0 / 1500: loss 41.763473
iteration 100 / 1500: loss 11.928966
iteration 200 / 1500: loss 9.260535
iteration 300 / 1500: loss 9.022207
iteration 400 / 1500: loss 9.001227
iteration 500 / 1500: loss 8.999098
iteration 600 / 1500: loss 8.999064
iteration 700 / 1500: loss 8.999171
iteration 800 / 1500: loss 8.999094
iteration 900 / 1500: loss 8.999110
iteration 1000 / 1500: loss 8.999224
iteration 1100 / 1500: loss 8.999188
iteration 1200 / 1500: loss 8.999206
iteration 1300 / 1500: loss 8.999165
iteration 1400 / 1500: loss 8.998876
iteration 0 / 1500: loss 57.657941
iteration 100 / 1500: loss 10.287605
iteration 200 / 1500: loss 9.033523
iteration 300 / 1500: loss 9.000453
iteration 400 / 1500: loss 8.999401
```

```
iteration 500 / 1500: loss 8.999362
iteration 600 / 1500: loss 8.999563
iteration 700 / 1500: loss 8.999465
iteration 800 / 1500: loss 8.999545
iteration 900 / 1500: loss 8.999398
iteration 1000 / 1500: loss 8.999354
iteration 1100 / 1500: loss 8.999484
iteration 1200 / 1500: loss 8.999383
iteration 1300 / 1500: loss 8.999403
iteration 1400 / 1500: loss 8.999410
iteration 0 / 1500: loss 75.892270
iteration 100 / 1500: loss 9.520302
iteration 200 / 1500: loss 9.003729
iteration 300 / 1500: loss 8.999623
iteration 400 / 1500: loss 8.999502
iteration 500 / 1500: loss 8.999656
iteration 600 / 1500: loss 8.999631
iteration 700 / 1500: loss 8.999546
iteration 800 / 1500: loss 8.999476
iteration 900 / 1500: loss 8.999507
iteration 1000 / 1500: loss 8.999578
iteration 1100 / 1500: loss 8.999559
iteration 1200 / 1500: loss 8.999521
iteration 1300 / 1500: loss 8.999553
iteration 1400 / 1500: loss 8.999527
iteration 0 / 1500: loss 89.518852
iteration 100 / 1500: loss 9.181799
iteration 200 / 1500: loss 9.000124
iteration 300 / 1500: loss 8.999649
iteration 400 / 1500: loss 8.999701
iteration 500 / 1500: loss 8.999619
iteration 600 / 1500: loss 8.999628
iteration 700 / 1500: loss 8.999640
iteration 800 / 1500: loss 8.999594
iteration 900 / 1500: loss 8.999611
iteration 1000 / 1500: loss 8.999734
iteration 1100 / 1500: loss 8.999569
iteration 1200 / 1500: loss 8.999629
iteration 1300 / 1500: loss 8.999596
iteration 1400 / 1500: loss 8.999583
iteration 0 / 1500: loss 161.811135
iteration 100 / 1500: loss 9.000467
iteration 200 / 1500: loss 8.999852
iteration 300 / 1500: loss 8.999852
iteration 400 / 1500: loss 8.999797
iteration 500 / 1500: loss 8.999783
iteration 600 / 1500: loss 8.999849
iteration 700 / 1500: loss 8.999827
iteration 800 / 1500: loss 8.999826
iteration 900 / 1500: loss 8.999832
iteration 1000 / 1500: loss 8.999842
iteration 1100 / 1500: loss 8.999863
iteration 1200 / 1500: loss 8.999851
```

```
iteration 1300 / 1500: loss 8.999799
iteration 1400 / 1500: loss 8.999831
iteration 0 / 1500: loss 10.729373
iteration 100 / 1500: loss 10.465829
iteration 200 / 1500: loss 10.264722
iteration 300 / 1500: loss 10.064632
iteration 400 / 1500: loss 9.909230
iteration 500 / 1500: loss 9.758948
iteration 600 / 1500: loss 9.651990
iteration 700 / 1500: loss 9.559395
iteration 800 / 1500: loss 9.472006
iteration 900 / 1500: loss 9.403993
iteration 1000 / 1500: loss 9.333529
iteration 1100 / 1500: loss 9.281952
iteration 1200 / 1500: loss 9.235019
iteration 1300 / 1500: loss 9.203810
iteration 1400 / 1500: loss 9.165482
iteration 0 / 1500: loss 17.327822
iteration 100 / 1500: loss 12.731024
iteration 200 / 1500: loss 10.664237
iteration 300 / 1500: loss 9.747534
iteration 400 / 1500: loss 9.337824
iteration 500 / 1500: loss 9.147404
iteration 600 / 1500: loss 9.063699
iteration 700 / 1500: loss 9.027401
iteration 800 / 1500: loss 9.010290
iteration 900 / 1500: loss 9.002423
iteration 1000 / 1500: loss 8.998465
iteration 1100 / 1500: loss 8.996756
iteration 1200 / 1500: loss 8.996777
iteration 1300 / 1500: loss 8.996360
iteration 1400 / 1500: loss 8.996326
iteration 0 / 1500: loss 24.396710
iteration 100 / 1500: loss 12.090010
iteration 200 / 1500: loss 9.618799
iteration 300 / 1500: loss 9.122673
iteration 400 / 1500: loss 9.023045
iteration 500 / 1500: loss 9.003485
iteration 600 / 1500: loss 8.999619
iteration 700 / 1500: loss 8.998247
iteration 800 / 1500: loss 8.998359
iteration 900 / 1500: loss 8.997749
iteration 1000 / 1500: loss 8.998128
iteration 1100 / 1500: loss 8.998368
iteration 1200 / 1500: loss 8.998521
iteration 1300 / 1500: loss 8.998490
iteration 1400 / 1500: loss 8.998483
iteration 0 / 1500: loss 42.884748
iteration 100 / 1500: loss 10.346457
iteration 200 / 1500: loss 9.052748
iteration 300 / 1500: loss 9.001272
iteration 400 / 1500: loss 8.999069
iteration 500 / 1500: loss 8.998997
```

```
iteration 600 / 1500: loss 8.999062
iteration 700 / 1500: loss 8.999167
iteration 800 / 1500: loss 8.999007
iteration 900 / 1500: loss 8.999224
iteration 1000 / 1500: loss 8.999115
iteration 1100 / 1500: loss 8.999282
iteration 1200 / 1500: loss 8.999243
iteration 1300 / 1500: loss 8.999141
iteration 1400 / 1500: loss 8.999122
iteration 0 / 1500: loss 57.042659
iteration 100 / 1500: loss 9.372103
iteration 200 / 1500: loss 9.002303
iteration 300 / 1500: loss 8.999517
iteration 400 / 1500: loss 8.999503
iteration 500 / 1500: loss 8.999315
iteration 600 / 1500: loss 8.999387
iteration 700 / 1500: loss 8.999357
iteration 800 / 1500: loss 8.999399
iteration 900 / 1500: loss 8.999391
iteration 1000 / 1500: loss 8.999494
iteration 1100 / 1500: loss 8.999353
iteration 1200 / 1500: loss 8.999349
iteration 1300 / 1500: loss 8.999480
iteration 1400 / 1500: loss 8.999298
iteration 0 / 1500: loss 72.398296
iteration 100 / 1500: loss 9.094401
iteration 200 / 1500: loss 8.999673
iteration 300 / 1500: loss 8.999496
iteration 400 / 1500: loss 8.999612
iteration 500 / 1500: loss 8.999604
iteration 600 / 1500: loss 8.999550
iteration 700 / 1500: loss 8.999600
iteration 800 / 1500: loss 8.999535
iteration 900 / 1500: loss 8.999512
iteration 1000 / 1500: loss 8.999676
iteration 1100 / 1500: loss 8.999496
iteration 1200 / 1500: loss 8.999633
iteration 1300 / 1500: loss 8.999523
iteration 1400 / 1500: loss 8.999590
iteration 0 / 1500: loss 90.342942
iteration 100 / 1500: loss 9.022753
iteration 200 / 1500: loss 8.999724
iteration 300 / 1500: loss 8.999726
iteration 400 / 1500: loss 8.999644
iteration 500 / 1500: loss 8.999647
iteration 600 / 1500: loss 8.999721
iteration 700 / 1500: loss 8.999716
iteration 800 / 1500: loss 8.999638
iteration 900 / 1500: loss 8.999684
iteration 1000 / 1500: loss 8.999610
iteration 1100 / 1500: loss 8.999708
iteration 1200 / 1500: loss 8.999627
iteration 1300 / 1500: loss 8.999707
```

```
iteration 1400 / 1500: loss 8.999655
iteration 0 / 1500: loss 173.556081
iteration 100 / 1500: loss 8.999862
iteration 200 / 1500: loss 8.999848
iteration 300 / 1500: loss 8.999832
iteration 400 / 1500: loss 8.999807
iteration 500 / 1500: loss 8.999828
iteration 600 / 1500: loss 8.999817
iteration 700 / 1500: loss 8.999855
iteration 800 / 1500: loss 8.999844
iteration 900 / 1500: loss 8.999846
iteration 1000 / 1500: loss 8.999830
iteration 1100 / 1500: loss 8.999879
iteration 1200 / 1500: loss 8.999808
iteration 1300 / 1500: loss 8.999812
iteration 1400 / 1500: loss 8.999814
iteration 0 / 1500: loss 10.570393
iteration 100 / 1500: loss 10.264311
iteration 200 / 1500: loss 10.047732
iteration 300 / 1500: loss 9.855832
iteration 400 / 1500: loss 9.684709
iteration 500 / 1500: loss 9.565026
iteration 600 / 1500: loss 9.451793
iteration 700 / 1500: loss 9.367290
iteration 800 / 1500: loss 9.297359
iteration 900 / 1500: loss 9.247584
iteration 1000 / 1500: loss 9.196811
iteration 1100 / 1500: loss 9.159311
iteration 1200 / 1500: loss 9.122963
iteration 1300 / 1500: loss 9.105164
iteration 1400 / 1500: loss 9.079747
iteration 0 / 1500: loss 16.981030
iteration 100 / 1500: loss 11.920799
iteration 200 / 1500: loss 10.069927
iteration 300 / 1500: loss 9.392087
iteration 400 / 1500: loss 9.143255
iteration 500 / 1500: loss 9.048390
iteration 600 / 1500: loss 9.015748
iteration 700 / 1500: loss 9.004352
iteration 800 / 1500: loss 8.999161
iteration 900 / 1500: loss 8.997283
iteration 1000 / 1500: loss 8.997040
iteration 1100 / 1500: loss 8.996720
iteration 1200 / 1500: loss 8.996523
iteration 1300 / 1500: loss 8.997189
iteration 1400 / 1500: loss 8.996433
iteration 0 / 1500: loss 25.307496
iteration 100 / 1500: loss 11.188444
iteration 200 / 1500: loss 9.293045
iteration 300 / 1500: loss 9.037573
iteration 400 / 1500: loss 9.003320
iteration 500 / 1500: loss 8.999025
iteration 600 / 1500: loss 8.998391
```

```
iteration 700 / 1500: loss 8.998195
iteration 800 / 1500: loss 8.998018
iteration 900 / 1500: loss 8.998440
iteration 1000 / 1500: loss 8.998252
iteration 1100 / 1500: loss 8.998526
iteration 1200 / 1500: loss 8.998471
iteration 1300 / 1500: loss 8.997971
iteration 1400 / 1500: loss 8.998179
iteration 0 / 1500: loss 41.368160
iteration 100 / 1500: loss 9.569448
iteration 200 / 1500: loss 9.008812
iteration 300 / 1500: loss 8.999039
iteration 400 / 1500: loss 8.999124
iteration 500 / 1500: loss 8.999072
iteration 600 / 1500: loss 8.999272
iteration 700 / 1500: loss 8.998984
iteration 800 / 1500: loss 8.999159
iteration 900 / 1500: loss 8.998962
iteration 1000 / 1500: loss 8.999010
iteration 1100 / 1500: loss 8.999078
iteration 1200 / 1500: loss 8.999084
iteration 1300 / 1500: loss 8.999181
iteration 1400 / 1500: loss 8.999083
iteration 0 / 1500: loss 54.815418
iteration 100 / 1500: loss 9.102831
iteration 200 / 1500: loss 8.999669
iteration 300 / 1500: loss 8.999447
iteration 400 / 1500: loss 8.999405
iteration 500 / 1500: loss 8.999452
iteration 600 / 1500: loss 8.999530
iteration 700 / 1500: loss 8.999408
iteration 800 / 1500: loss 8.999338
iteration 900 / 1500: loss 8.999496
iteration 1000 / 1500: loss 8.999509
iteration 1100 / 1500: loss 8.999203
iteration 1200 / 1500: loss 8.999437
iteration 1300 / 1500: loss 8.999345
iteration 1400 / 1500: loss 8.999330
iteration 0 / 1500: loss 75.362154
iteration 100 / 1500: loss 9.018300
iteration 200 / 1500: loss 8.999580
iteration 300 / 1500: loss 8.999650
iteration 400 / 1500: loss 8.999458
iteration 500 / 1500: loss 8.999642
iteration 600 / 1500: loss 8.999487
iteration 700 / 1500: loss 8.999515
iteration 800 / 1500: loss 8.999649
iteration 900 / 1500: loss 8.999542
iteration 1000 / 1500: loss 8.999501
iteration 1100 / 1500: loss 8.999602
iteration 1200 / 1500: loss 8.999582
iteration 1300 / 1500: loss 8.999585
iteration 1400 / 1500: loss 8.999560
```

```
iteration 0 / 1500: loss 86.802750
iteration 100 / 1500: loss 9.002371
iteration 200 / 1500: loss 8.999649
iteration 300 / 1500: loss 8.999634
iteration 400 / 1500: loss 8.999653
iteration 500 / 1500: loss 8.999657
iteration 600 / 1500: loss 8.999642
iteration 700 / 1500: loss 8.999632
iteration 800 / 1500: loss 8.999622
iteration 900 / 1500: loss 8.999620
iteration 1000 / 1500: loss 8.999635
iteration 1100 / 1500: loss 8.999676
iteration 1200 / 1500: loss 8.999669
iteration 1300 / 1500: loss 8.999626
iteration 1400 / 1500: loss 8.999655
iteration 0 / 1500: loss 161.531972
iteration 100 / 1500: loss 8.999815
iteration 200 / 1500: loss 8.999822
iteration 300 / 1500: loss 8.999835
iteration 400 / 1500: loss 8.999802
iteration 500 / 1500: loss 8.999814
iteration 600 / 1500: loss 8.999784
iteration 700 / 1500: loss 8.999858
iteration 800 / 1500: loss 8.999833
iteration 900 / 1500: loss 8.999802
iteration 1000 / 1500: loss 8.999793
iteration 1100 / 1500: loss 8.999823
iteration 1200 / 1500: loss 8.999889
iteration 1300 / 1500: loss 8.999873
iteration 1400 / 1500: loss 8.999832
iteration 0 / 1500: loss 10.587828
iteration 100 / 1500: loss 10.052891
iteration 200 / 1500: loss 9.702093
iteration 300 / 1500: loss 9.458461
iteration 400 / 1500: loss 9.306641
iteration 500 / 1500: loss 9.196640
iteration 600 / 1500: loss 9.128164
iteration 700 / 1500: loss 9.076964
iteration 800 / 1500: loss 9.044735
iteration 900 / 1500: loss 9.026516
iteration 1000 / 1500: loss 9.010755
iteration 1100 / 1500: loss 9.001037
iteration 1200 / 1500: loss 8.994874
iteration 1300 / 1500: loss 8.989765
iteration 1400 / 1500: loss 8.989583
iteration 0 / 1500: loss 16.568270
iteration 100 / 1500: loss 10.011221
iteration 200 / 1500: loss 9.133680
iteration 300 / 1500: loss 9.013916
iteration 400 / 1500: loss 8.998662
iteration 500 / 1500: loss 8.996388
iteration 600 / 1500: loss 8.996028
iteration 700 / 1500: loss 8.996395
```

```
iteration 800 / 1500: loss 8.996658
iteration 900 / 1500: loss 8.995615
iteration 1000 / 1500: loss 8.996330
iteration 1100 / 1500: loss 8.996738
iteration 1200 / 1500: loss 8.996399
iteration 1300 / 1500: loss 8.996673
iteration 1400 / 1500: loss 8.996325
iteration 0 / 1500: loss 24.854267
iteration 100 / 1500: loss 9.278244
iteration 200 / 1500: loss 9.002978
iteration 300 / 1500: loss 8.998335
iteration 400 / 1500: loss 8.998720
iteration 500 / 1500: loss 8.998517
iteration 600 / 1500: loss 8.997834
iteration 700 / 1500: loss 8.998193
iteration 800 / 1500: loss 8.997918
iteration 900 / 1500: loss 8.998207
iteration 1000 / 1500: loss 8.998369
iteration 1100 / 1500: loss 8.997978
iteration 1200 / 1500: loss 8.998069
iteration 1300 / 1500: loss 8.998181
iteration 1400 / 1500: loss 8.998478
iteration 0 / 1500: loss 41.654656
iteration 100 / 1500: loss 9.008239
iteration 200 / 1500: loss 8.998992
iteration 300 / 1500: loss 8.998919
iteration 400 / 1500: loss 8.999109
iteration 500 / 1500: loss 8.999208
iteration 600 / 1500: loss 8.999118
iteration 700 / 1500: loss 8.998965
iteration 800 / 1500: loss 8.999188
iteration 900 / 1500: loss 8.998938
iteration 1000 / 1500: loss 8.999082
iteration 1100 / 1500: loss 8.999238
iteration 1200 / 1500: loss 8.999082
iteration 1300 / 1500: loss 8.999304
iteration 1400 / 1500: loss 8.999151
iteration 0 / 1500: loss 58.108084
iteration 100 / 1500: loss 8.999538
iteration 200 / 1500: loss 8.999393
iteration 300 / 1500: loss 8.999497
iteration 400 / 1500: loss 8.999308
iteration 500 / 1500: loss 8.999375
iteration 600 / 1500: loss 8.999478
iteration 700 / 1500: loss 8.999545
iteration 800 / 1500: loss 8.999515
iteration 900 / 1500: loss 8.999319
iteration 1000 / 1500: loss 8.999495
iteration 1100 / 1500: loss 8.999382
iteration 1200 / 1500: loss 8.999423
iteration 1300 / 1500: loss 8.999396
iteration 1400 / 1500: loss 8.999458
iteration 0 / 1500: loss 75.523520
```

```
iteration 100 / 1500: loss 8.999494
iteration 200 / 1500: loss 8.999553
iteration 300 / 1500: loss 8.999547
iteration 400 / 1500: loss 8.999507
iteration 500 / 1500: loss 8.999614
iteration 600 / 1500: loss 8.999510
iteration 700 / 1500: loss 8.999666
iteration 800 / 1500: loss 8.999523
iteration 900 / 1500: loss 8.999602
iteration 1000 / 1500: loss 8.999673
iteration 1100 / 1500: loss 8.999549
iteration 1200 / 1500: loss 8.999615
iteration 1300 / 1500: loss 8.999557
iteration 1400 / 1500: loss 8.999568
iteration 0 / 1500: loss 85.872054
iteration 100 / 1500: loss 8.999647
iteration 200 / 1500: loss 8.999685
iteration 300 / 1500: loss 8.999686
iteration 400 / 1500: loss 8.999697
iteration 500 / 1500: loss 8.999716
iteration 600 / 1500: loss 8.999745
iteration 700 / 1500: loss 8.999635
iteration 800 / 1500: loss 8.999690
iteration 900 / 1500: loss 8.999684
iteration 1000 / 1500: loss 8.999669
iteration 1100 / 1500: loss 8.999610
iteration 1200 / 1500: loss 8.999605
iteration 1300 / 1500: loss 8.999643
iteration 1400 / 1500: loss 8.999704
iteration 0 / 1500: loss 173.709683
iteration 100 / 1500: loss 8.999843
iteration 200 / 1500: loss 8.999824
iteration 300 / 1500: loss 8.999836
iteration 400 / 1500: loss 8.999850
iteration 500 / 1500: loss 8.999813
iteration 600 / 1500: loss 8.999864
iteration 700 / 1500: loss 8.999886
iteration 800 / 1500: loss 8.999812
iteration 900 / 1500: loss 8.999853
iteration 1000 / 1500: loss 8.999815
iteration 1100 / 1500: loss 8.999854
iteration 1200 / 1500: loss 8.999830
iteration 1300 / 1500: loss 8.999827
iteration 1400 / 1500: loss 8.999812
iteration 0 / 1500: loss 10.644058
iteration 100 / 1500: loss 9.207834
iteration 200 / 1500: loss 9.011109
iteration 300 / 1500: loss 8.988933
iteration 400 / 1500: loss 8.984772
iteration 500 / 1500: loss 8.983945
iteration 600 / 1500: loss 8.983209
iteration 700 / 1500: loss 8.980983
iteration 800 / 1500: loss 8.981477
```

```
iteration 900 / 1500: loss 8.981896
iteration 1000 / 1500: loss 8.979910
iteration 1100 / 1500: loss 8.983869
iteration 1200 / 1500: loss 8.979598
iteration 1300 / 1500: loss 8.980929
iteration 1400 / 1500: loss 8.983668
iteration 0 / 1500: loss 17.379961
iteration 100 / 1500: loss 8.996374
iteration 200 / 1500: loss 8.995983
iteration 300 / 1500: loss 8.995851
iteration 400 / 1500: loss 8.997152
iteration 500 / 1500: loss 8.996199
iteration 600 / 1500: loss 8.996209
iteration 700 / 1500: loss 8.995864
iteration 800 / 1500: loss 8.995865
iteration 900 / 1500: loss 8.996624
iteration 1000 / 1500: loss 8.996295
iteration 1100 / 1500: loss 8.995846
iteration 1200 / 1500: loss 8.996987
iteration 1300 / 1500: loss 8.996483
iteration 1400 / 1500: loss 8.996271
iteration 0 / 1500: loss 25.867173
iteration 100 / 1500: loss 8.998509
iteration 200 / 1500: loss 8.998708
iteration 300 / 1500: loss 8.998210
iteration 400 / 1500: loss 8.998283
iteration 500 / 1500: loss 8.998032
iteration 600 / 1500: loss 8.998144
iteration 700 / 1500: loss 8.998215
iteration 800 / 1500: loss 8.998542
iteration 900 / 1500: loss 8.998573
iteration 1000 / 1500: loss 8.997999
iteration 1100 / 1500: loss 8.998693
iteration 1200 / 1500: loss 8.998112
iteration 1300 / 1500: loss 8.998575
iteration 1400 / 1500: loss 8.997898
iteration 0 / 1500: loss 42.014623
iteration 100 / 1500: loss 8.999297
iteration 200 / 1500: loss 8.998963
iteration 300 / 1500: loss 8.999306
iteration 400 / 1500: loss 8.999105
iteration 500 / 1500: loss 8.999178
iteration 600 / 1500: loss 8.999129
iteration 700 / 1500: loss 8.999370
iteration 800 / 1500: loss 8.999198
iteration 900 / 1500: loss 8.999138
iteration 1000 / 1500: loss 8.999369
iteration 1100 / 1500: loss 8.999068
iteration 1200 / 1500: loss 8.999063
iteration 1300 / 1500: loss 8.999171
iteration 1400 / 1500: loss 8.999220
iteration 0 / 1500: loss 57.341200
iteration 100 / 1500: loss 8.999549
```

```
iteration 200 / 1500: loss 8.999563
iteration 300 / 1500: loss 8.999709
iteration 400 / 1500: loss 8.999482
iteration 500 / 1500: loss 8.999421
iteration 600 / 1500: loss 8.999578
iteration 700 / 1500: loss 8.999588
iteration 800 / 1500: loss 8.999414
iteration 900 / 1500: loss 8.999363
iteration 1000 / 1500: loss 8.999493
iteration 1100 / 1500: loss 8.999473
iteration 1200 / 1500: loss 8.999486
iteration 1300 / 1500: loss 8.999490
iteration 1400 / 1500: loss 8.999658
iteration 0 / 1500: loss 74.177490
iteration 100 / 1500: loss 8.999791
iteration 200 / 1500: loss 8.999668
iteration 300 / 1500: loss 8.999729
iteration 400 / 1500: loss 8.999738
iteration 500 / 1500: loss 8.999627
iteration 600 / 1500: loss 8.999656
iteration 700 / 1500: loss 8.999694
iteration 800 / 1500: loss 8.999696
iteration 900 / 1500: loss 8.999695
iteration 1000 / 1500: loss 8.999639
iteration 1100 / 1500: loss 8.999769
iteration 1200 / 1500: loss 8.999578
iteration 1300 / 1500: loss 8.999685
iteration 1400 / 1500: loss 8.999714
iteration 0 / 1500: loss 93.935979
iteration 100 / 1500: loss 8.999766
iteration 200 / 1500: loss 8.999712
iteration 300 / 1500: loss 8.999835
iteration 400 / 1500: loss 8.999750
iteration 500 / 1500: loss 8.999812
iteration 600 / 1500: loss 8.999726
iteration 700 / 1500: loss 8.999796
iteration 800 / 1500: loss 8.999877
iteration 900 / 1500: loss 8.999825
iteration 1000 / 1500: loss 8.999736
iteration 1100 / 1500: loss 8.999741
iteration 1200 / 1500: loss 8.999696
iteration 1300 / 1500: loss 8.999707
iteration 1400 / 1500: loss 8.999687
iteration 0 / 1500: loss 172.675160
iteration 100 / 1500: loss 9.000009
iteration 200 / 1500: loss 8.999969
iteration 300 / 1500: loss 9.000000
iteration 400 / 1500: loss 8.999967
iteration 500 / 1500: loss 8.999976
iteration 600 / 1500: loss 8.999971
iteration 700 / 1500: loss 9.000044
iteration 800 / 1500: loss 8.999956
iteration 900 / 1500: loss 8.999927
```

```
iteration 1000 / 1500: loss 9.000006
iteration 1100 / 1500: loss 8.999998
iteration 1200 / 1500: loss 9.000052
iteration 1300 / 1500: loss 9.000013
iteration 1400 / 1500: loss 9.000010
lr 5.000000e-08 reg 1.000000e+03 train accuracy: 0.098429 val accura
cy: 0.098000
lr 5.000000e-08 reg 5.000000e+03 train accuracy: 0.113735 val accura
cy: 0.117000
lr 5.000000e-08 reg 1.000000e+04 train accuracy: 0.132571 val accura
cy: 0.123000
lr 5.000000e-08 reg 2.000000e+04 train accuracy: 0.177735 val accura
cy: 0.192000
lr 5.000000e-08 reg 3.000000e+04 train accuracy: 0.246490 val accura
cy: 0.239000
lr 5.000000e-08 reg 4.000000e+04 train accuracy: 0.382469 val accura
cy: 0.400000
lr 5.000000e-08 reg 5.000000e+04 train accuracy: 0.416327 val accura
cy: 0.408000
lr 5.000000e-08 reg 1.000000e+05 train accuracy: 0.417878 val accura
cy: 0.422000
lr 1.000000e-07 reg 1.000000e+03 train accuracy: 0.123122 val accura
cy: 0.134000
lr 1.000000e-07 reg 5.000000e+03 train accuracy: 0.151082 val accura
cy: 0.147000
lr 1.000000e-07 reg 1.000000e+04 train accuracy: 0.216490 val accura
cy: 0.228000
lr 1.000000e-07 reg 2.000000e+04 train accuracy: 0.405327 val accura
cy: 0.409000
lr 1.000000e-07 reg 3.000000e+04 train accuracy: 0.417612 val accura
cy: 0.419000
lr 1.000000e-07 reg 4.000000e+04 train accuracy: 0.422959 val accura
cy: 0.420000
lr 1.000000e-07 reg 5.000000e+04 train accuracy: 0.417020 val accura
cy: 0.426000
lr 1.000000e-07 reg 1.000000e+05 train accuracy: 0.423184 val accura
cy: 0.435000
lr 2.000000e-07 reg 1.000000e+03 train accuracy: 0.146694 val accura
cy: 0.144000
lr 2.000000e-07 reg 5.000000e+03 train accuracy: 0.291020 val accura
cy: 0.305000
lr 2.000000e-07 reg 1.000000e+04 train accuracy: 0.415959 val accura
cy: 0.406000
lr 2.000000e-07 reg 2.000000e+04 train accuracy: 0.417551 val accura
cy: 0.428000
lr 2.000000e-07 reg 3.000000e+04 train accuracy: 0.419061 val accura
cy: 0.438000
lr 2.000000e-07 reg 4.000000e+04 train accuracy: 0.416653 val accura
cy: 0.412000
lr 2.000000e-07 reg 5.000000e+04 train accuracy: 0.419898 val accura
cy: 0.420000
lr 2.000000e-07 reg 1.000000e+05 train accuracy: 0.412796 val accura
cy: 0.404000
```

```
lr 3.000000e-07 reg 1.000000e+03 train accuracy: 0.205796 val accura
cy: 0.216000
lr 3.000000e-07 reg 5.000000e+03 train accuracy: 0.397898 val accura
cy: 0.393000
lr 3.000000e-07 reg 1.000000e+04 train accuracy: 0.418143 val accura
cy: 0.426000
lr 3.000000e-07 reg 2.000000e+04 train accuracy: 0.419245 val accura
cy: 0.421000
lr 3.000000e-07 reg 3.000000e+04 train accuracy: 0.420061 val accura
cy: 0.429000
lr 3.000000e-07 reg 4.000000e+04 train accuracy: 0.414653 val accura
cy: 0.420000
lr 3.000000e-07 reg 5.000000e+04 train accuracy: 0.412490 val accura
cy: 0.413000
lr 3.000000e-07 reg 1.000000e+05 train accuracy: 0.414673 val accura
cy: 0.422000
lr 4.000000e-07 reg 1.000000e+03 train accuracy: 0.246531 val accura
cy: 0.249000
lr 4.000000e-07 reg 5.000000e+03 train accuracy: 0.417490 val accura
cy: 0.422000
lr 4.000000e-07 reg 1.000000e+04 train accuracy: 0.418898 val accura
cy: 0.421000
lr 4.000000e-07 reg 2.000000e+04 train accuracy: 0.417347 val accura
cy: 0.421000
lr 4.000000e-07 reg 3.000000e+04 train accuracy: 0.416531 val accura
cy: 0.410000
lr 4.000000e-07 reg 4.000000e+04 train accuracy: 0.414816 val accura
cy: 0.412000
lr 4.000000e-07 reg 5.000000e+04 train accuracy: 0.408816 val accura
cy: 0.404000
lr 4.000000e-07 reg 1.000000e+05 train accuracy: 0.420000 val accura
cy: 0.415000
lr 5.000000e-07 reg 1.000000e+03 train accuracy: 0.261265 val accura
cy: 0.303000
lr 5.000000e-07 reg 5.000000e+03 train accuracy: 0.416816 val accura
cy: 0.425000
lr 5.000000e-07 reg 1.000000e+04 train accuracy: 0.420408 val accura
cy: 0.432000
lr 5.000000e-07 reg 2.000000e+04 train accuracy: 0.412306 val accura
cy: 0.414000
lr 5.000000e-07 reg 3.000000e+04 train accuracy: 0.416204 val accura
cy: 0.407000
lr 5.000000e-07 reg 4.000000e+04 train accuracy: 0.417796 val accura
cy: 0.417000
lr 5.000000e-07 reg 5.000000e+04 train accuracy: 0.414102 val accura
cy: 0.418000
lr 5.000000e-07 reg 1.000000e+05 train accuracy: 0.407510 val accura
cy: 0.405000
lr 1.000000e-06 reg 1.000000e+03 train accuracy: 0.408980 val accura
cy: 0.401000
lr 1.000000e-06 reg 5.000000e+03 train accuracy: 0.416551 val accura
cy: 0.415000
lr 1.000000e-06 reg 1.000000e+04 train accuracy: 0.417694 val accura
```

```
cy: 0.412000
lr 1.000000e-06 reg 2.000000e+04 train accuracy: 0.415306 val accura
cy: 0.412000
lr 1.000000e-06 reg 3.000000e+04 train accuracy: 0.419918 val accura
cy: 0.420000
lr 1.000000e-06 reg 4.000000e+04 train accuracy: 0.407918 val accura
cy: 0.404000
lr 1.000000e-06 reg 5.000000e+04 train accuracy: 0.415796 val accura
cy: 0.411000
lr 1.000000e-06 reg 1.000000e+05 train accuracy: 0.405612 val accura
cy: 0.410000
lr 5.000000e-06 reg 1.000000e+03 train accuracy: 0.420694 val accura
cy: 0.421000
lr 5.000000e-06 reg 5.000000e+03 train accuracy: 0.411082 val accura
cy: 0.416000
lr 5.000000e-06 reg 1.000000e+04 train accuracy: 0.412694 val accura
cy: 0.416000
lr 5.000000e-06 reg 2.000000e+04 train accuracy: 0.411633 val accura
cy: 0.413000
lr 5.000000e-06 reg 3.000000e+04 train accuracy: 0.398939 val accura
cy: 0.408000
lr 5.000000e-06 reg 4.000000e+04 train accuracy: 0.364837 val accura
cy: 0.356000
lr 5.000000e-06 reg 5.000000e+04 train accuracy: 0.369918 val accura
cy: 0.374000
lr 5.000000e-06 reg 1.000000e+05 train accuracy: 0.325653 val accura
cy: 0.325000
best validation accuracy achieved during cross-validation: 0.438000
```

In [14]:

```
# 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)
```

```
0.426
```

In [15]:

```
# 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?

Some seem to make sense - for example, peripheral view of bird that can arguably look like a plane taking off or a dog on a grassy field standing on 4 legs mistaken for a horse. But many other misclassifications are hard to interpret.

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 [29]:

```
# 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, 161)
(49000, 160)
```

In [33]:

```
from cs231n.classifiers.neural_net import TwoLayerNet

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

################################################################################
# TODO: Train a two-layer neural network on image features. You may want to    #
# cross-validate various parameters as in previous sections. Store your best   #
# model in the best_net variable.                                              #
################################################################################
results = {}
best_val = -1
best_net = None
best_params = None
learning_rates = [1e-1, 5e-1, 1e0]
regularization_strengths = [1e-3, 5e-3, 1e-2]

for l in learning_rates:
    for r in regularization_strengths:
        print(l)
        net = TwoLayerNet(input_dim, hidden_dim, num_classes)
        curr_loss = net.train(X_train_feats, y_train, X_val_feats, y_val, learni
ng_rate=l, reg=r, num_iters=1500)
        y_train_pred = net.predict(X_train_feats)
        y_train_acc = np.mean(y_train == y_train_pred)
        y_val_pred = net.predict(X_val_feats)
        y_val_acc = np.mean(y_val == y_val_pred)
        results[(l, r)] = (y_train_acc, y_val_acc)
        if y_val_acc > best_val:
            best_net = net
            best_val = y_val_acc
            best_params = (l, r)


################################################################################
#                              END OF YOUR CODE                                #
################################################################################
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
)
print(best_params)
```

```
0.1
0.1
0.1
0.5
0.5
0.5
1.0
1.0
1.0
lr 1.000000e-01 reg 1.000000e-03 train accuracy: 0.540918 val accura
cy: 0.525000
lr 1.000000e-01 reg 5.000000e-03 train accuracy: 0.528388 val accura
cy: 0.514000
lr 1.000000e-01 reg 1.000000e-02 train accuracy: 0.521020 val accura
cy: 0.520000
lr 5.000000e-01 reg 1.000000e-03 train accuracy: 0.660694 val accura
cy: 0.597000
lr 5.000000e-01 reg 5.000000e-03 train accuracy: 0.565143 val accura
cy: 0.559000
lr 5.000000e-01 reg 1.000000e-02 train accuracy: 0.514714 val accura
cy: 0.500000
lr 1.000000e+00 reg 1.000000e-03 train accuracy: 0.671020 val accura
cy: 0.560000
lr 1.000000e+00 reg 5.000000e-03 train accuracy: 0.555163 val accura
cy: 0.519000
lr 1.000000e+00 reg 1.000000e-02 train accuracy: 0.509469 val accura
cy: 0.486000
best validation accuracy achieved during cross-validation: 0.597000
(0.5, 0.001)
```

In [34]:

```
# 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)
```

```
0.577
```