knn

August 6, 2021

[6]:
```
# This mounts your Google Drive to the Colab VM.
from google.colab import drive
drive.mount('/content/drive', force_remount=True)

# Enter the foldername in your Drive where you have saved the unzipped
# assignment folder, e.g. 'cs231n/assignments/assignment1/'
FOLDERNAME = 'CS231N/assignment1/'
assert FOLDERNAME is not None, "[!] Enter the foldername."

# Now that we've mounted your Drive, this ensures that
# the Python interpreter of the Colab VM can load
# python files from within it.
import sys
sys.path.append('/content/drive/My Drive/{}'.format(FOLDERNAME))

# This downloads the CIFAR-10 dataset to your Drive
# if it doesn't already exist.
%cd drive/My\ Drive/$FOLDERNAME/cs231n/datasets/
!bash get_datasets.sh
%cd /content/drive/My\ Drive/$FOLDERNAME
```

```
Mounted at /content/drive
/content/drive/My Drive/CS231N/assignment1/cs231n/datasets
/content/drive/My Drive/CS231N/assignment1
```

1 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 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

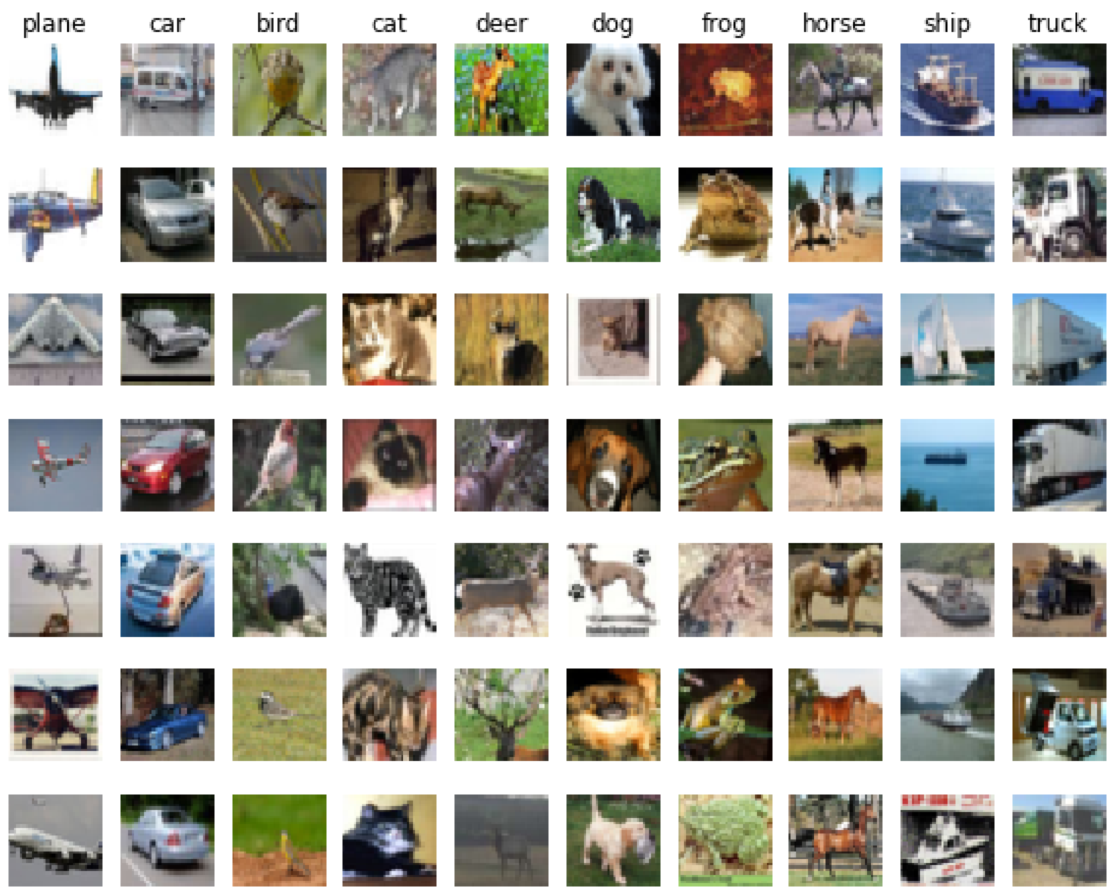
In this exercise you will implement these steps and understand the basic Image Classification pipeline, cross-validation, and gain proficiency in writing efficient, vectorized code.

[7]:
```
# Run some setup code for this notebook.

import random
import numpy as np
from cs231n.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 plots
plt.rcParams['image.interpolation'] = 'nearest'
plt.rcParams['image.cmap'] = 'gray'

# Some more magic so that the notebook will reload external python modules;
# see http://stackoverflow.com/questions/1907993/
 ↪autoreload-of-modules-in-ipython
%reload_ext autoreload
%autoreload 2
```

[8]:
```
# Load the raw CIFAR-10 data.
cifar10_dir = 'cs231n/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 data.
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)
```

```
Training data shape:  (50000, 32, 32, 3)
Training labels shape:  (50000,)
Test data shape:  (10000, 32, 32, 3)
Test labels shape:  (10000,)
```

[9]:
```
# 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', 'horse',␣
 ↪'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()
```



[10]:
```
# Subsample the data for more efficient code execution in this exercise
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]

# 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)
```

[11]:
```
from cs231n.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 processing
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.

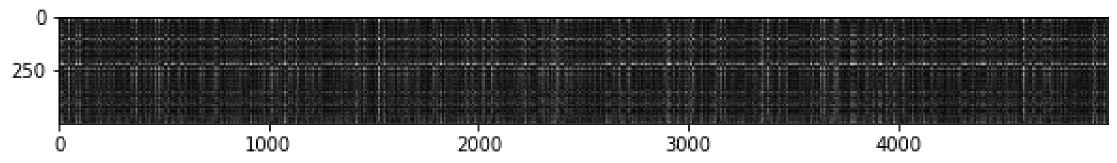
Note: For the three distance computations that we require you to implement in this notebook, you may not use the np.linalg.norm() function that numpy provides.

First, open `cs231n/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.

[12]:
```
# Open cs231n/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)
```

[13]:
```python
# We can visualize the distance matrix: each row is a single test example 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?

$Your Answer:$ A bright row occur when a picture in the test set has high distance with almost every picture in the train set. However, a bright column occur when a picture in the train set has high distance with almost every picture in the test set.

[14]:
```python
# 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`:

[15]:
```python
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 138 / 500 correct => accuracy: 0.276000
```

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

Inline Question 2

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

the mean μ across all pixels over all images is

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

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

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

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

Which of the following preprocessing steps will not change the performance of a Nearest Neighbor classifier that uses L1 distance? Select all that apply. 1. Subtracting the mean μ ($\tilde{p}_{ij}^{(k)} = p_{ij}^{(k)} - \mu$.) 2. Subtracting the per pixel mean μ_{ij} ($\tilde{p}_{ij}^{(k)} = p_{ij}^{(k)} - \mu_{ij}$.) 3. Subtracting the mean μ and dividing by the standard deviation σ. 4. Subtracting the pixel-wise mean μ_{ij} and dividing by the pixel-wise standard deviation σ_{ij}. 5. Rotating the coordinate axes of the data.

YourAnswer : 1, 2, 3

YourExplanation : L1 distance can be calculated as below:

$$\sum_{i=1}^{h} \sum_{j=1}^{w} |\tilde{p}_{ij}^{(k)} - \tilde{q}_{ij}^{(l)}|$$

When we subtract the mean or the pixel mean for p and q, they are cancelled out, and thus doesn't change the result. Also, dividing by constant value σ is a simple re-scaling of the result and does not change the nearest neighbor. However, dividing by pixel-wise standard deviations will change the result.

[16]:
```
# Now lets speed up distance matrix computation by using partial vectorization
# with one loop. Implement the function compute_distances_one_loop and run the
# code below:
dists_one = classifier.compute_distances_one_loop(X_test)

# To ensure that our vectorized implementation is correct, we make sure 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 norm. In case
# you haven't seen it before, the Frobenius norm of two matrices is the square
# root of the squared sum of differences of all elements; in other words,␣
 ↪reshape
# the matrices into vectors and compute the Euclidean distance between them.
difference = np.linalg.norm(dists - dists_one, ord='fro')
print('One loop 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')
```

```
One loop difference was: 0.000000
Good! The distance matrices are the same
```

[17]:
```python
# 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 before:
difference = np.linalg.norm(dists - dists_two, ord='fro')
print('No loop 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')
```

```
No loop difference was: 0.000000
Good! The distance matrices are the same
```

[18]:
```python
# 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 vectorized␣
 ↪implementation!

# NOTE: depending on what machine you're using,
# you might not see a speedup when you go from two loops to one loop,
# and might even see a slow-down.
```

```
Two loop version took 17.032274 seconds
One loop version took 18.098101 seconds
No loop version took 1.787601 seconds
```

1.0.1 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.

[31]:
```
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_folds 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_folds[i].     ␣
 ↪#
# Hint: Look up the numpy array_split function.                                ␣
 ↪#
################################################################################
# *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****

for i in np.array_split(X_train, num_folds):
  X_train_folds.append(i)
for j in np.array_split(y_train, num_folds):
  y_train_folds.append(j)

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

# 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 each        ␣
 ↪#
# 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 data and the␣
 ↪#
```

```
# last fold as a validation set. Store the accuracies for all fold and all ␣
↪#
# values of k in the k_to_accuracies dictionary.                            ␣
↪#
################################################################################
# *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****

for kval in k_choices:

  #print("now processing kval = {}".format(kval))

  accuracies = []

  for i in range(kval):

    X_train_set = np.concatenate([x for j, x in enumerate(X_train_folds) if i!
↪=j])
    y_train_set = np.concatenate([x for j, x in enumerate(y_train_folds) if i!
↪=j])

    classifier.train(X_train_set, y_train_set)
    y_test_pred = classifier.predict_labels(dists, k=kval)
    num_correct = np.sum(y_test_pred == y_test)
    acc = float(num_correct) / num_test

    accuracies.append(acc)

  k_to_accuracies[kval] = accuracies

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

# 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 = 0.136000
k = 3, accuracy = 0.124000
k = 3, accuracy = 0.180000
k = 3, accuracy = 0.214000
k = 5, accuracy = 0.138000
k = 5, accuracy = 0.178000
k = 5, accuracy = 0.222000
k = 5, accuracy = 0.256000
k = 5, accuracy = 0.258000
k = 8, accuracy = 0.150000
k = 8, accuracy = 0.176000
```

```
k = 8, accuracy = 0.230000
k = 8, accuracy = 0.244000
k = 8, accuracy = 0.240000
k = 8, accuracy = 0.240000
k = 8, accuracy = 0.240000
k = 8, accuracy = 0.240000
k = 10, accuracy = 0.144000
k = 10, accuracy = 0.220000
k = 10, accuracy = 0.222000
k = 10, accuracy = 0.266000
k = 10, accuracy = 0.240000
k = 10, accuracy = 0.240000
k = 10, accuracy = 0.240000
k = 10, accuracy = 0.240000
k = 10, accuracy = 0.240000
k = 10, accuracy = 0.240000
k = 12, accuracy = 0.174000
k = 12, accuracy = 0.214000
k = 12, accuracy = 0.230000
k = 12, accuracy = 0.256000
k = 12, accuracy = 0.258000
k = 12, accuracy = 0.258000
k = 12, accuracy = 0.258000
k = 12, accuracy = 0.258000
k = 12, accuracy = 0.258000
k = 12, accuracy = 0.258000
k = 12, accuracy = 0.258000
k = 12, accuracy = 0.258000
k = 15, accuracy = 0.192000
k = 15, accuracy = 0.220000
k = 15, accuracy = 0.218000
k = 15, accuracy = 0.264000
k = 15, accuracy = 0.254000
k = 15, accuracy = 0.254000
k = 15, accuracy = 0.254000
k = 15, accuracy = 0.254000
k = 15, accuracy = 0.254000
k = 15, accuracy = 0.254000
k = 15, accuracy = 0.254000
k = 15, accuracy = 0.254000
k = 15, accuracy = 0.254000
k = 15, accuracy = 0.254000
k = 15, accuracy = 0.254000
k = 20, accuracy = 0.186000
k = 20, accuracy = 0.222000
k = 20, accuracy = 0.236000
k = 20, accuracy = 0.254000
k = 20, accuracy = 0.270000
```

```
k = 20, accuracy = 0.270000
k = 20, accuracy = 0.270000
k = 20, accuracy = 0.270000
k = 20, accuracy = 0.270000
k = 20, accuracy = 0.270000
k = 20, accuracy = 0.270000
k = 20, accuracy = 0.270000
k = 20, accuracy = 0.270000
k = 20, accuracy = 0.270000
k = 20, accuracy = 0.270000
k = 20, accuracy = 0.270000
k = 20, accuracy = 0.270000
k = 20, accuracy = 0.270000
k = 20, accuracy = 0.270000
k = 20, accuracy = 0.270000
k = 50, accuracy = 0.202000
k = 50, accuracy = 0.222000
k = 50, accuracy = 0.232000
k = 50, accuracy = 0.242000
k = 50, accuracy = 0.232000
k = 50, accuracy = 0.232000
k = 50, accuracy = 0.232000
k = 50, accuracy = 0.232000
k = 50, accuracy = 0.232000
k = 50, accuracy = 0.232000
k = 50, accuracy = 0.232000
k = 50, accuracy = 0.232000
k = 50, accuracy = 0.232000
k = 50, accuracy = 0.232000
k = 50, accuracy = 0.232000
k = 50, accuracy = 0.232000
k = 50, accuracy = 0.232000
k = 50, accuracy = 0.232000
k = 50, accuracy = 0.232000
k = 50, accuracy = 0.232000
k = 50, accuracy = 0.232000
k = 50, accuracy = 0.232000
k = 50, accuracy = 0.232000
k = 50, accuracy = 0.232000
k = 50, accuracy = 0.232000
k = 50, accuracy = 0.232000
k = 50, accuracy = 0.232000
k = 50, accuracy = 0.232000
k = 50, accuracy = 0.232000
k = 50, accuracy = 0.232000
k = 50, accuracy = 0.232000
k = 50, accuracy = 0.232000
```

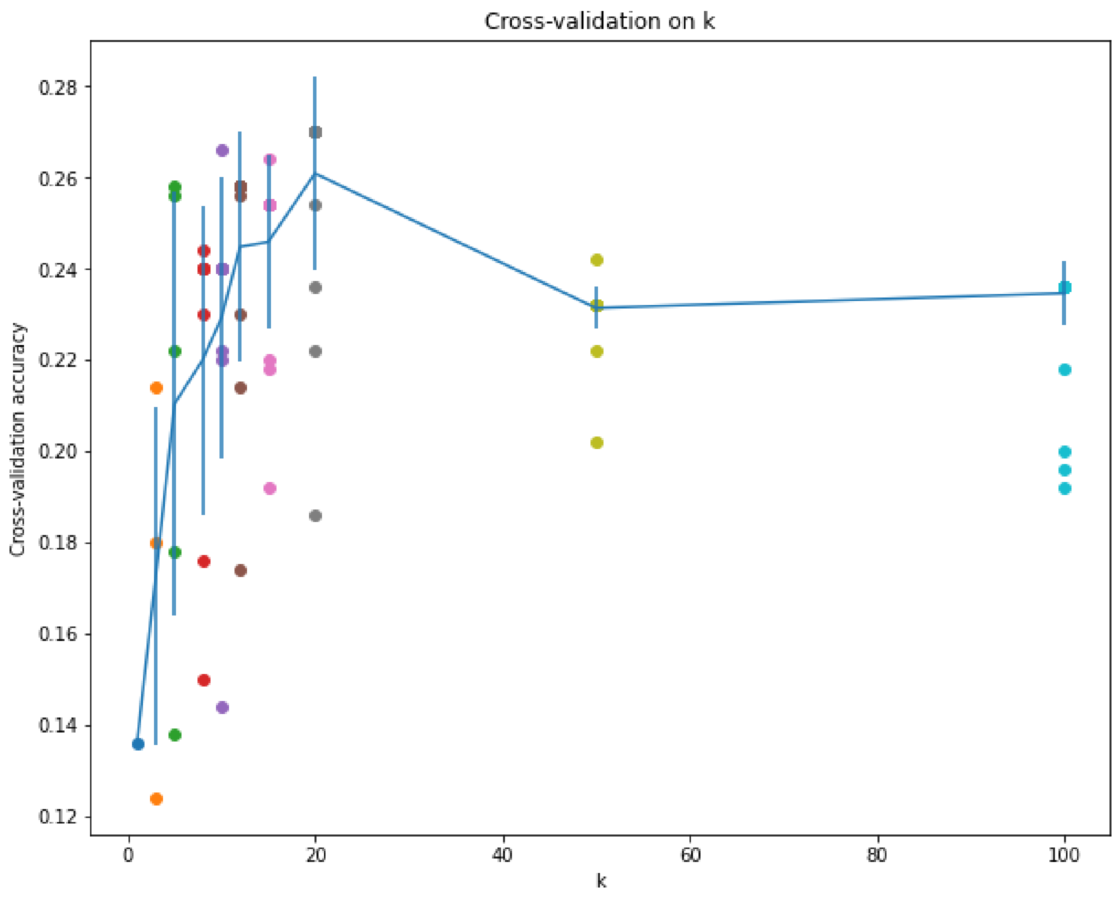
```
k = 50, accuracy = 0.232000
k = 50, accuracy = 0.232000
k = 50, accuracy = 0.232000
k = 50, accuracy = 0.232000
k = 50, accuracy = 0.232000
k = 50, accuracy = 0.232000
k = 50, accuracy = 0.232000
k = 50, accuracy = 0.232000
k = 50, accuracy = 0.232000
k = 50, accuracy = 0.232000
k = 50, accuracy = 0.232000
k = 50, accuracy = 0.232000
k = 50, accuracy = 0.232000
k = 50, accuracy = 0.232000
k = 50, accuracy = 0.232000
k = 50, accuracy = 0.232000
k = 50, accuracy = 0.232000
k = 100, accuracy = 0.192000
k = 100, accuracy = 0.196000
k = 100, accuracy = 0.200000
k = 100, accuracy = 0.218000
k = 100, accuracy = 0.236000
k = 100, accuracy = 0.236000
k = 100, accuracy = 0.236000
k = 100, accuracy = 0.236000
k = 100, accuracy = 0.236000
k = 100, accuracy = 0.236000
k = 100, accuracy = 0.236000
k = 100, accuracy = 0.236000
k = 100, accuracy = 0.236000
k = 100, accuracy = 0.236000
k = 100, accuracy = 0.236000
k = 100, accuracy = 0.236000
k = 100, accuracy = 0.236000
k = 100, accuracy = 0.236000
k = 100, accuracy = 0.236000
k = 100, accuracy = 0.236000
k = 100, accuracy = 0.236000
k = 100, accuracy = 0.236000
k = 100, accuracy = 0.236000
k = 100, accuracy = 0.236000
k = 100, accuracy = 0.236000
k = 100, accuracy = 0.236000
k = 100, accuracy = 0.236000
k = 100, accuracy = 0.236000
k = 100, accuracy = 0.236000
k = 100, accuracy = 0.236000
k = 100, accuracy = 0.236000
```

```
k = 100, accuracy = 0.236000
k = 100, accuracy = 0.236000
k = 100, accuracy = 0.236000
k = 100, accuracy = 0.236000
k = 100, accuracy = 0.236000
k = 100, accuracy = 0.236000
k = 100, accuracy = 0.236000
k = 100, accuracy = 0.236000
k = 100, accuracy = 0.236000
k = 100, accuracy = 0.236000
k = 100, accuracy = 0.236000
k = 100, accuracy = 0.236000
k = 100, accuracy = 0.236000
k = 100, accuracy = 0.236000
k = 100, accuracy = 0.236000
k = 100, accuracy = 0.236000
k = 100, accuracy = 0.236000
k = 100, accuracy = 0.236000
k = 100, accuracy = 0.236000
k = 100, accuracy = 0.236000
k = 100, accuracy = 0.236000
k = 100, accuracy = 0.236000
k = 100, accuracy = 0.236000
k = 100, accuracy = 0.236000
k = 100, accuracy = 0.236000
k = 100, accuracy = 0.236000
k = 100, accuracy = 0.236000
k = 100, accuracy = 0.236000
k = 100, accuracy = 0.236000
k = 100, accuracy = 0.236000
k = 100, accuracy = 0.236000
k = 100, accuracy = 0.236000
k = 100, accuracy = 0.236000
k = 100, accuracy = 0.236000
k = 100, accuracy = 0.236000
k = 100, accuracy = 0.236000
k = 100, accuracy = 0.236000
k = 100, accuracy = 0.236000
k = 100, accuracy = 0.236000
k = 100, accuracy = 0.236000
k = 100, accuracy = 0.236000
k = 100, accuracy = 0.236000
k = 100, accuracy = 0.236000
k = 100, accuracy = 0.236000
k = 100, accuracy = 0.236000
k = 100, accuracy = 0.236000
k = 100, accuracy = 0.236000
k = 100, accuracy = 0.236000
```

```
k = 100, accuracy = 0.236000
k = 100, accuracy = 0.236000
k = 100, accuracy = 0.236000
k = 100, accuracy = 0.236000
k = 100, accuracy = 0.236000
k = 100, accuracy = 0.236000
k = 100, accuracy = 0.236000
k = 100, accuracy = 0.236000
k = 100, accuracy = 0.236000
k = 100, accuracy = 0.236000
k = 100, accuracy = 0.236000
k = 100, accuracy = 0.236000
k = 100, accuracy = 0.236000
k = 100, accuracy = 0.236000
k = 100, accuracy = 0.236000
k = 100, accuracy = 0.236000
k = 100, accuracy = 0.236000
k = 100, accuracy = 0.236000
k = 100, accuracy = 0.236000
k = 100, accuracy = 0.236000
k = 100, accuracy = 0.236000
```

[22]:

```
# 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 deviation
accuracies_mean = np.array([np.mean(v) for k,v in sorted(k_to_accuracies.
 ↪items())])
accuracies_std = np.array([np.std(v) for k,v in sorted(k_to_accuracies.
 ↪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()
```



[30]:
```
# 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 data.
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 decision boundary of the k-NN classifier is linear. 2. The training error of a 1-NN will always be lower than or equal to that of 5-NN. 3. The test error of a 1-NN will always be lower than that of a 5-NN. 4. The time needed to classify a test example with the k-NN classifier grows with the size of the training set. 5. None of the above.

$YourAnswer:$ 2, 4

$YourExplanation:$ When 1-NN is applied, training error should be 0 since the identical data can be found in the training set and hence the distance will always be 0. This is not the case when 5-NN is applied. When the size of the training set of the k-NN classifier becomes larger, the cost needed to calculate the distances increases.

svm

August 6, 2021

[1]:
```
# This mounts your Google Drive to the Colab VM.
from google.colab import drive
drive.mount('/content/drive', force_remount=True)

# Enter the foldername in your Drive where you have saved the unzipped
# assignment folder, e.g. 'cs231n/assignments/assignment1/'
FOLDERNAME = 'CS231N/assignment1/'
assert FOLDERNAME is not None, "[!] Enter the foldername."

# Now that we've mounted your Drive, this ensures that
# the Python interpreter of the Colab VM can load
# python files from within it.
import sys
sys.path.append('/content/drive/My Drive/{}'.format(FOLDERNAME))

# This downloads the CIFAR-10 dataset to your Drive
# if it doesn't already exist.
%cd drive/My\ Drive/$FOLDERNAME/cs231n/datasets/
!bash get_datasets.sh
%cd /content/drive/My\ Drive/$FOLDERNAME
```

```
Mounted at /content/drive
/content/drive/My Drive/CS231N/assignment1/cs231n/datasets
/content/drive/My Drive/CS231N/assignment1
```

1 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 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

[11]:
```
# Run some setup code for this notebook.
import random
import numpy as np
from cs231n.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 plots
plt.rcParams['image.interpolation'] = 'nearest'
plt.rcParams['image.cmap'] = 'gray'

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

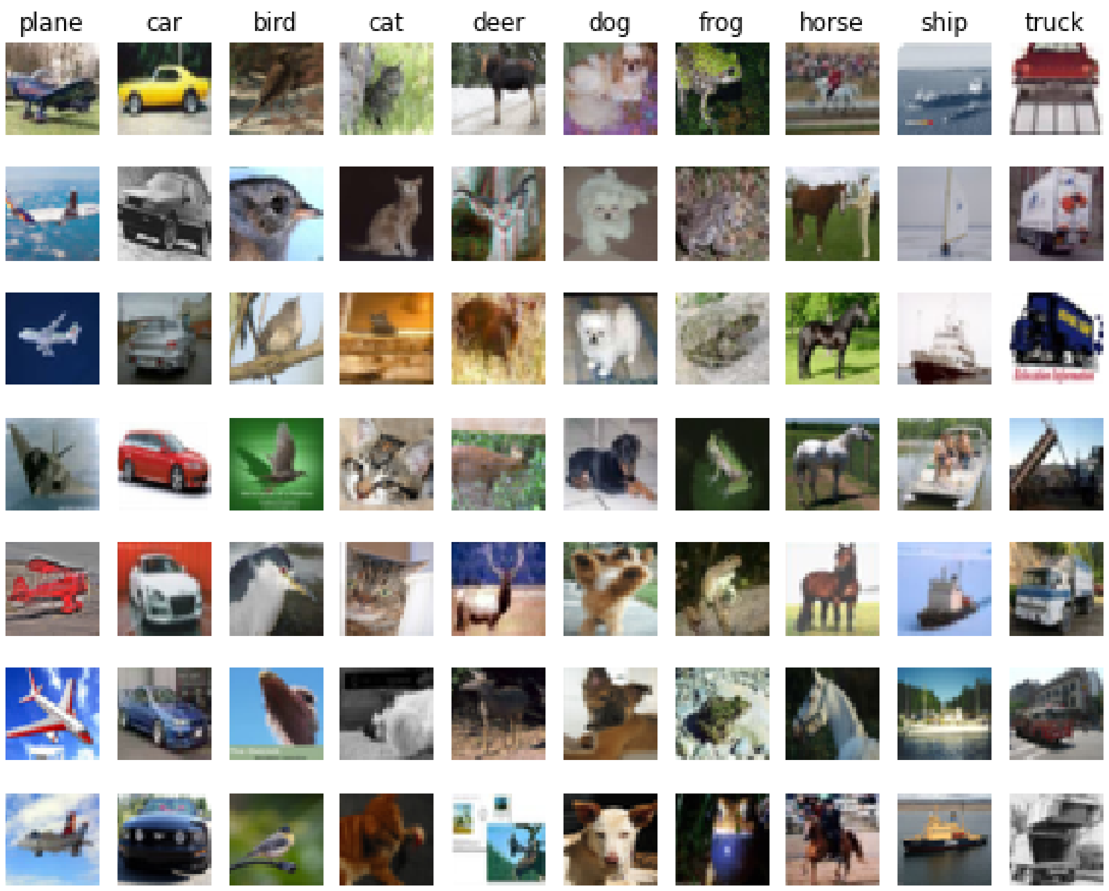
```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
```

1.1 CIFAR-10 Data Loading and Preprocessing

[12]:
```
# Load the raw CIFAR-10 data.
cifar10_dir = 'cs231n/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 data.
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,)
```

[13]:
```
# 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', 'horse',␣
 ↪'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()
```



[14]:
```
# 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 original
# 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,)
```

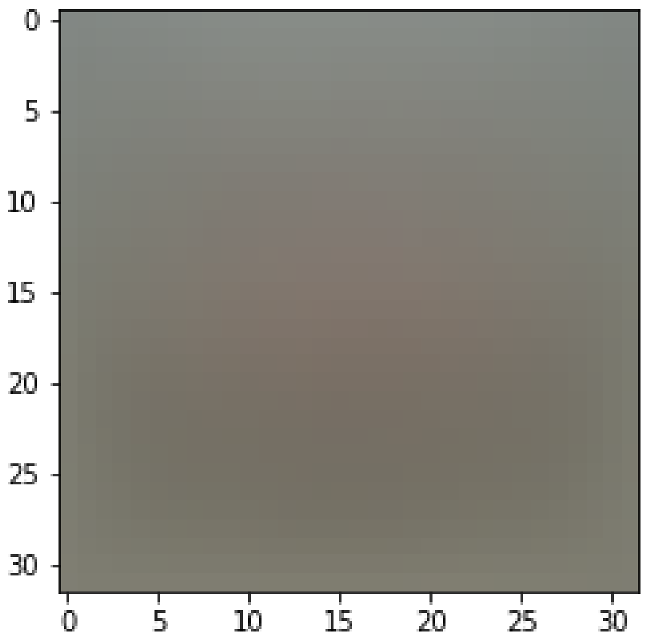
[15]:
```
# Preprocessing: reshape the image data into rows
X_train = np.reshape(X_train, (X_train.shape[0], -1))
X_val = np.reshape(X_val, (X_val.shape[0], -1))
X_test = np.reshape(X_test, (X_test.shape[0], -1))
X_dev = np.reshape(X_dev, (X_dev.shape[0], -1))

# 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)
```

[16]:
```
# 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()

# 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

# 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)
```

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



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

1.2 SVM Classifier

Your code for this section will all be written inside `cs231n/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.

[17]:
```
# Evaluate the naive implementation of the loss we provided for you:
from cs231n.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: 8.912287
```

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:

[18]:
```
# Once you've implemented the gradient, recompute it with the code below
# and gradient check it with the function we provided for you

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

# Numerically compute the gradient along several randomly chosen dimensions,␣
 ↪and
# compare them with your analytically computed gradient. The numbers should␣
 ↪match
# almost exactly along all dimensions.
from cs231n.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: 21.207294 analytic: 21.207294, relative error: 2.340802e-12
numerical: 14.429357 analytic: 14.429357, relative error: 1.535694e-11
numerical: -34.101114 analytic: -34.101114, relative error: 9.313848e-12
```

```
numerical: -6.017741 analytic: -6.017741, relative error: 2.101080e-11
numerical: -28.906145 analytic: -28.853978, relative error: 9.031550e-04
numerical: -1.194958 analytic: -1.194958, relative error: 3.544871e-10
numerical: -2.638236 analytic: -2.638236, relative error: 7.872470e-11
numerical: 13.644781 analytic: 13.621929, relative error: 8.380742e-04
numerical: 22.226167 analytic: 22.226167, relative error: 2.227423e-12
numerical: -11.215754 analytic: -11.215754, relative error: 2.474228e-11
numerical: 22.948344 analytic: 22.948344, relative error: 7.386927e-13
numerical: 14.841339 analytic: 14.841339, relative error: 2.365046e-11
numerical: -42.128462 analytic: -42.128462, relative error: 8.504561e-12
numerical: 21.417958 analytic: 21.351004, relative error: 1.565478e-03
numerical: 16.550601 analytic: 16.550601, relative error: 1.599683e-11
numerical: 4.227711 analytic: 4.227711, relative error: 1.747980e-11
numerical: -19.918116 analytic: -19.918116, relative error: 1.182477e-11
numerical: 2.695495 analytic: 2.695495, relative error: 9.874510e-11
numerical: -27.274681 analytic: -27.274681, relative error: 3.092297e-13
numerical: -33.975730 analytic: -33.975730, relative error: 5.173832e-12
```

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*

$YourAnswer:$ Assume L(x) = max(x, 0). Analytically, L(x) should only have gradient between 0 or 1. However, the numerical gradient calculated with the interval including x=0 will return the value which is not equal to 0 nor 1. In other words, a discrepancy occurs at non-differentiable points. The change of the margin would not affect the frequency of the occurence of the discrepancy, since the number of non-differential points is not changed.

[19]:
```
# Next implement the function svm_loss_vectorized; for now only compute 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 cs231n.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 be much␣
 ↪faster.
print('difference: %f' % (loss_naive - loss_vectorized))
```

```
Naive loss: 8.912287e+00 computed in 0.125609s
```

```
Vectorized loss: 8.912287e+00 computed in 0.014108s
difference: 0.000000
```

[20]:
```python
# 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 match, 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 computed
# by the two implementations. The gradient on the other hand is a matrix, 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 0.153174s
Vectorized loss and gradient: computed in 0.011894s
difference: 0.000000
```

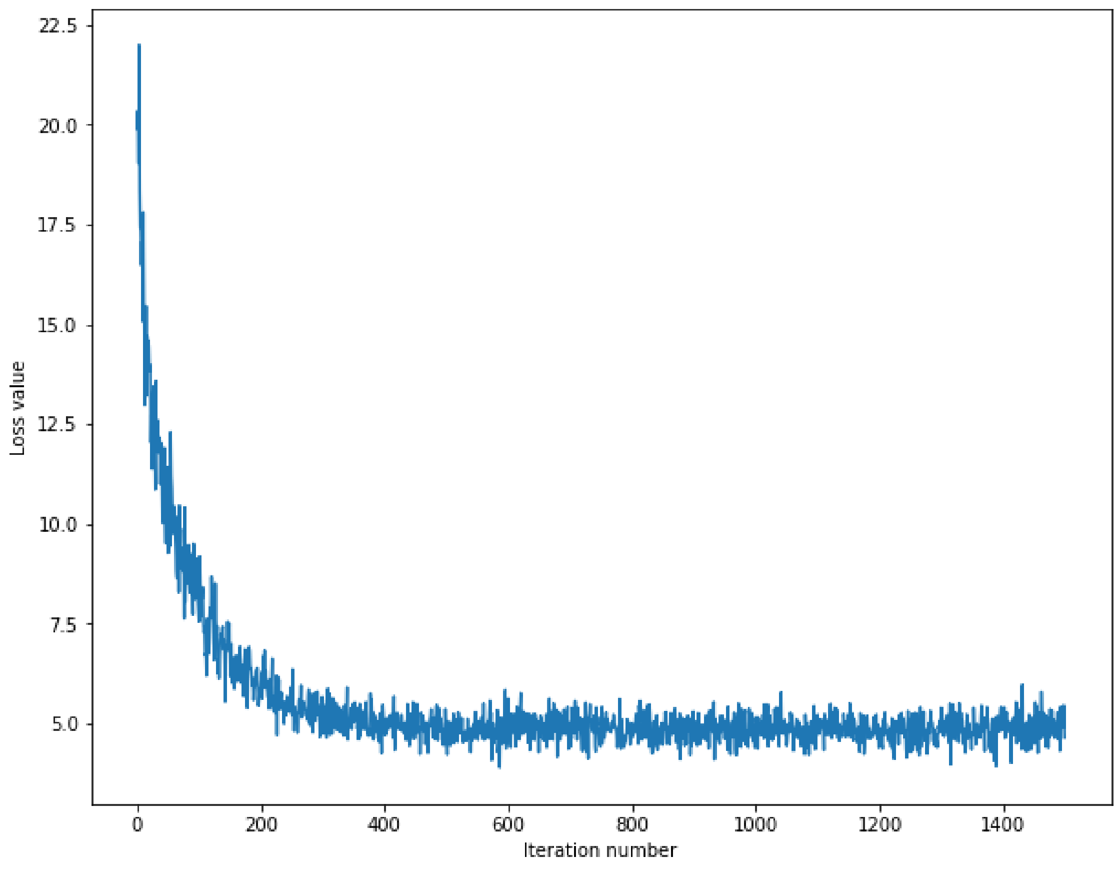
1.2.1 Stochastic Gradient Descent

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. Your code for this part will be written inside `cs231n/classifiers/linear_classifier.py`.

[47]:
```python
# In the file linear_classifier.py, implement SGD in the function
# LinearClassifier.train() and then run it with the code below.
from cs231n.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=1500, verbose=True)
toc = time.time()
print('That took %fs' % (toc - tic))
```

```
iteration 0 / 1500: loss 19.888843
iteration 100 / 1500: loss 7.551138
iteration 200 / 1500: loss 6.030363
iteration 300 / 1500: loss 5.558220
iteration 400 / 1500: loss 4.725600
iteration 500 / 1500: loss 4.572640
iteration 600 / 1500: loss 5.635151
iteration 700 / 1500: loss 4.423327
iteration 800 / 1500: loss 5.021391
iteration 900 / 1500: loss 4.565024
iteration 1000 / 1500: loss 5.158611
iteration 1100 / 1500: loss 5.231441
iteration 1200 / 1500: loss 4.684391
iteration 1300 / 1500: loss 5.401669
iteration 1400 / 1500: loss 5.001369
That took 10.153024s
```

[48]:
```
# 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()
```



[43]:
```python
# Write the LinearSVM.predict function and evaluate the performance on 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: 0.364918
validation accuracy: 0.382000
```

[70]:
```python
# Use the validation set to tune hyperparameters (regularization strength and
# learning rate). You should experiment with different ranges for the learning
# rates and regularization strengths; if you are careful you should be able to
# get a classification accuracy of about 0.39 on the validation set.

# Note: you may see runtime/overflow warnings during hyper-parameter search.
# This may be caused by extreme values, and is not a bug.

# 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 the fraction
# of data points that are correctly classified.
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.

################################################################################
# TODO:                                                                       ␣
 ↪#
# Write code that chooses the best hyperparameters by tuning on the validation␣
 ↪#
# set. For each combination of hyperparameters, train a linear SVM on the     ␣
 ↪#
# training set, compute its accuracy on the training and validation sets, and ␣
 ↪#
# store these numbers in the results dictionary. In addition, store the best  ␣
 ↪#
# validation accuracy in best_val and the LinearSVM object that achieves this ␣
 ↪#
# accuracy in best_svm.                                                       ␣
 ↪#
#                                                                             ␣
 ↪#
# Hint: You should use a small value for num_iters as you develop your        ␣
 ↪#
```

```
# validation code so that the SVMs don't take much time to train; once you are␣
↪#
# confident that your validation code works, you should rerun the validation ␣
↪#
# code with a larger value for num_iters.                                    ␣
↪#
################################################################################

# Provided as a reference. You may or may not want to change these␣
↪hyperparameters
learning_rates = np.linspace(5e-8, 5e-7, 50)
regularization_strengths = np.linspace(2.5e4, 5e4, 10)

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

max_count = 300
for count in range(max_count):
  print("count {} / {}".format(count, max_count))
  lr = 10**np.random.uniform(-8, -4)
  rs = 10**np.random.uniform(3, 5)

  test_svm = LinearSVM()
  test_loss = test_svm.train(X_train, y_train, learning_rate=lr, reg=rs,
                    num_iters=20, verbose=True)
  if test_loss[-1] < 12.5:
    real_svm = LinearSVM()
    real_svm.train(X_train, y_train, learning_rate=lr, reg=rs,
                    num_iters=1000, verbose=True)

    y_val_pred = real_svm.predict(X_val)
    val_accuracy = np.mean(y_val == y_val_pred)
    y_train_pred = real_svm.predict(X_train)
    train_accuracy = np.mean(y_train == y_train_pred)
    results[(lr, rs)] = (train_accuracy, val_accuracy)

    if val_accuracy > best_val:
      best_val = val_accuracy
      best_svm = real_svm
      print("best_val updated to %f" %val_accuracy)

    else:
      print("Val_accuracy: {}, Current best_val: {}".format(val_accuracy,␣
↪best_val))

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

# 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 5.344701e-07 reg 1.262751e+03 train accuracy: 0.379306 val accuracy: 0.
 ↪400000
```

```
count 0 / 300
iteration 0 / 20: loss 19.920132
count 1 / 300
iteration 0 / 20: loss 21.975083
count 2 / 300
iteration 0 / 20: loss 24.591784
count 3 / 300
iteration 0 / 20: loss 20.705548
count 4 / 300
iteration 0 / 20: loss 28.372435
count 5 / 300
iteration 0 / 20: loss 17.376777
count 6 / 300
iteration 0 / 20: loss 23.265607
count 7 / 300
iteration 0 / 20: loss 18.953271
count 8 / 300
iteration 0 / 20: loss 23.669043
count 9 / 300
iteration 0 / 20: loss 23.921342
count 10 / 300
iteration 0 / 20: loss 20.276476
count 11 / 300
iteration 0 / 20: loss 22.495127
count 12 / 300
iteration 0 / 20: loss 24.759255
count 13 / 300
iteration 0 / 20: loss 18.609118
iteration 0 / 1000: loss 23.407444
iteration 100 / 1000: loss 5.045416
iteration 200 / 1000: loss 5.924994
iteration 300 / 1000: loss 5.401434
iteration 400 / 1000: loss 4.830746
iteration 500 / 1000: loss 5.441053
iteration 600 / 1000: loss 5.488769
iteration 700 / 1000: loss 5.677912
iteration 800 / 1000: loss 5.259326
iteration 900 / 1000: loss 6.013626
best_val updated to 0.350000
count 14 / 300
iteration 0 / 20: loss 22.080882
count 15 / 300
iteration 0 / 20: loss 28.617870
iteration 0 / 1000: loss 21.974451
iteration 100 / 1000: loss 7.456239
iteration 200 / 1000: loss 7.051494
iteration 300 / 1000: loss 5.996995
iteration 400 / 1000: loss 5.654096
iteration 500 / 1000: loss 8.367532
iteration 600 / 1000: loss 6.634850
iteration 700 / 1000: loss 5.858057
iteration 800 / 1000: loss 5.722877
iteration 900 / 1000: loss 5.323946
Val_accuracy: 0.313, Current best_val: 0.35
count 16 / 300
iteration 0 / 20: loss 23.637705
count 17 / 300
iteration 0 / 20: loss 25.486686
count 18 / 300
iteration 0 / 20: loss 23.584694
count 19 / 300
iteration 0 / 20: loss 23.287353
count 20 / 300
iteration 0 / 20: loss 20.938496
iteration 0 / 1000: loss 33.341673
iteration 100 / 1000: loss 7.169929
iteration 200 / 1000: loss 7.056913
iteration 300 / 1000: loss 5.411458
iteration 400 / 1000: loss 5.442535
iteration 500 / 1000: loss 4.874461
iteration 600 / 1000: loss 4.956423
iteration 700 / 1000: loss 4.497339
iteration 800 / 1000: loss 5.169805
iteration 900 / 1000: loss 4.821526
best_val updated to 0.358000
count 21 / 300
iteration 0 / 20: loss 24.502372
iteration 0 / 1000: loss 23.762104
iteration 100 / 1000: loss 8.818364
iteration 200 / 1000: loss 9.254569
iteration 300 / 1000: loss 7.660952
iteration 400 / 1000: loss 7.892684
iteration 500 / 1000: loss 7.239308
iteration 600 / 1000: loss 9.734692
iteration 700 / 1000: loss 8.452973
iteration 800 / 1000: loss 9.197865
iteration 900 / 1000: loss 7.493977
Val_accuracy: 0.223, Current best_val: 0.358
count 22 / 300
iteration 0 / 20: loss 21.398177
count 23 / 300
iteration 0 / 20: loss 25.365025
count 24 / 300
iteration 0 / 20: loss 23.192052
count 25 / 300
iteration 0 / 20: loss 21.204462
count 26 / 300
iteration 0 / 20: loss 20.928407
count 27 / 300
iteration 0 / 20: loss 23.751826
count 28 / 300
iteration 0 / 20: loss 19.645712
count 29 / 300
iteration 0 / 20: loss 18.928454
iteration 0 / 1000: loss 18.840925
iteration 100 / 1000: loss 5.840163
iteration 200 / 1000: loss 6.169744
iteration 300 / 1000: loss 6.395415
iteration 400 / 1000: loss 6.525966
iteration 500 / 1000: loss 6.497672
iteration 600 / 1000: loss 6.485129
iteration 700 / 1000: loss 6.246658
iteration 800 / 1000: loss 7.612745
iteration 900 / 1000: loss 6.853458
Val_accuracy: 0.291, Current best_val: 0.358
count 30 / 300
iteration 0 / 20: loss 21.151217
iteration 0 / 1000: loss 24.762892
iteration 100 / 1000: loss 7.015039
iteration 200 / 1000: loss 7.904226
iteration 300 / 1000: loss 7.280512
iteration 400 / 1000: loss 8.642968
iteration 500 / 1000: loss 4.934989
iteration 600 / 1000: loss 5.973305
iteration 700 / 1000: loss 5.507908
iteration 800 / 1000: loss 7.935033
iteration 900 / 1000: loss 6.605000
Val_accuracy: 0.266, Current best_val: 0.358
count 31 / 300
iteration 0 / 20: loss 23.423467
count 32 / 300
iteration 0 / 20: loss 18.555339
iteration 0 / 1000: loss 22.693767
iteration 100 / 1000: loss 5.539849
iteration 200 / 1000: loss 4.609645
iteration 300 / 1000: loss 5.269991
iteration 400 / 1000: loss 5.225231
iteration 500 / 1000: loss 4.887929
iteration 600 / 1000: loss 4.771280
iteration 700 / 1000: loss 5.160574
iteration 800 / 1000: loss 5.022711
iteration 900 / 1000: loss 4.700155
best_val updated to 0.370000
count 33 / 300
iteration 0 / 20: loss 22.095485
iteration 0 / 1000: loss 22.063376
iteration 100 / 1000: loss 6.133604
iteration 200 / 1000: loss 4.651273
iteration 300 / 1000: loss 4.287973
iteration 400 / 1000: loss 4.425412
iteration 500 / 1000: loss 5.080003
iteration 600 / 1000: loss 5.290558
iteration 700 / 1000: loss 5.427009
iteration 800 / 1000: loss 5.650836
iteration 900 / 1000: loss 5.289092
Val_accuracy: 0.318, Current best_val: 0.37
count 34 / 300
iteration 0 / 20: loss 22.130779
count 35 / 300
iteration 0 / 20: loss 23.340778
iteration 0 / 1000: loss 23.256732
iteration 100 / 1000: loss 5.348273
iteration 200 / 1000: loss 6.032541
iteration 300 / 1000: loss 5.870072
iteration 400 / 1000: loss 5.307243
iteration 500 / 1000: loss 5.648365
iteration 600 / 1000: loss 5.188811
iteration 700 / 1000: loss 5.300332
iteration 800 / 1000: loss 6.892902
iteration 900 / 1000: loss 5.745283
Val_accuracy: 0.328, Current best_val: 0.37
count 36 / 300
iteration 0 / 20: loss 22.579618
count 37 / 300
iteration 0 / 20: loss 21.345986
iteration 0 / 1000: loss 20.792033
iteration 100 / 1000: loss 5.023190
iteration 200 / 1000: loss 4.473006
iteration 300 / 1000: loss 5.537566
iteration 400 / 1000: loss 5.379884
iteration 500 / 1000: loss 4.947181
iteration 600 / 1000: loss 5.047189
iteration 700 / 1000: loss 5.011934
iteration 800 / 1000: loss 5.359225
iteration 900 / 1000: loss 5.109769
Val_accuracy: 0.356, Current best_val: 0.37
count 38 / 300
iteration 0 / 20: loss 24.078649
iteration 0 / 1000: loss 18.652018
iteration 100 / 1000: loss 6.071781
iteration 200 / 1000: loss 5.471874
iteration 300 / 1000: loss 5.455301
iteration 400 / 1000: loss 6.159824
iteration 500 / 1000: loss 5.261726
iteration 600 / 1000: loss 6.652937
iteration 700 / 1000: loss 6.092021
iteration 800 / 1000: loss 7.071062
iteration 900 / 1000: loss 6.304326
Val_accuracy: 0.287, Current best_val: 0.37
count 39 / 300
iteration 0 / 20: loss 22.136836
count 40 / 300
iteration 0 / 20: loss 25.599720
count 41 / 300
iteration 0 / 20: loss 20.937690
count 42 / 300
iteration 0 / 20: loss 23.086162
count 43 / 300
iteration 0 / 20: loss 20.329494
count 44 / 300
iteration 0 / 20: loss 25.562566
iteration 0 / 1000: loss 24.062303
iteration 100 / 1000: loss 6.087199
iteration 200 / 1000: loss 5.726860
iteration 300 / 1000: loss 4.970944
iteration 400 / 1000: loss 4.837601
iteration 500 / 1000: loss 5.176774
iteration 600 / 1000: loss 5.732265
iteration 700 / 1000: loss 5.256538
iteration 800 / 1000: loss 4.706303
iteration 900 / 1000: loss 4.446568
best_val updated to 0.385000
count 45 / 300
iteration 0 / 20: loss 18.678022
count 46 / 300
iteration 0 / 20: loss 18.387713
count 47 / 300
iteration 0 / 20: loss 20.801707
count 48 / 300
```

```
iteration 0 / 20: loss 20.318647
count 49 / 300
iteration 0 / 20: loss 20.833265
iteration 0 / 1000: loss 20.306910
iteration 100 / 1000: loss 6.131651
iteration 200 / 1000: loss 5.007107
iteration 300 / 1000: loss 5.716891
iteration 400 / 1000: loss 5.224804
iteration 500 / 1000: loss 4.623084
iteration 600 / 1000: loss 4.464085
iteration 700 / 1000: loss 5.040471
iteration 800 / 1000: loss 5.049542
iteration 900 / 1000: loss 5.285475
Val_accuracy: 0.369, Current best_val: 0.385
count 50 / 300
iteration 0 / 20: loss 21.513746
count 51 / 300
iteration 0 / 20: loss 25.940759
count 52 / 300
iteration 0 / 20: loss 21.351825
iteration 0 / 1000: loss 22.449801
iteration 100 / 1000: loss 10.543467
iteration 200 / 1000: loss 12.536034
iteration 300 / 1000: loss 8.437843
iteration 400 / 1000: loss 11.335512
iteration 500 / 1000: loss 8.692373
iteration 600 / 1000: loss 11.011041
iteration 700 / 1000: loss 10.905193
iteration 800 / 1000: loss 12.451294
iteration 900 / 1000: loss 12.618633
Val_accuracy: 0.245, Current best_val: 0.385
count 53 / 300
iteration 0 / 20: loss 22.441606
iteration 0 / 1000: loss 17.467958
iteration 100 / 1000: loss 7.332817
iteration 200 / 1000: loss 5.651743
iteration 300 / 1000: loss 5.030063
iteration 400 / 1000: loss 4.332079
iteration 500 / 1000: loss 4.883645
iteration 600 / 1000: loss 4.308993
iteration 700 / 1000: loss 5.248830
iteration 800 / 1000: loss 5.322141
iteration 900 / 1000: loss 4.252091
Val_accuracy: 0.375, Current best_val: 0.385
count 54 / 300
iteration 0 / 20: loss 24.587174
iteration 0 / 1000: loss 21.751175
iteration 100 / 1000: loss 6.020435
iteration 200 / 1000: loss 5.113001
iteration 300 / 1000: loss 4.537905
iteration 400 / 1000: loss 4.828519
iteration 500 / 1000: loss 4.197777
iteration 600 / 1000: loss 5.476647
iteration 700 / 1000: loss 4.770100
iteration 800 / 1000: loss 4.654681
iteration 900 / 1000: loss 4.999039
Val_accuracy: 0.366, Current best_val: 0.385
count 55 / 300
iteration 0 / 20: loss 25.368894
count 56 / 300
iteration 0 / 20: loss 23.090287
count 57 / 300
iteration 0 / 20: loss 24.105385
count 58 / 300
iteration 0 / 20: loss 20.468619
count 59 / 300
iteration 0 / 20: loss 22.361242
count 60 / 300
iteration 0 / 20: loss 23.257454
count 61 / 300
iteration 0 / 20: loss 20.635346
count 62 / 300
iteration 0 / 20: loss 27.087964
count 63 / 300
iteration 0 / 20: loss 21.351822
count 64 / 300
iteration 0 / 20: loss 19.876174
count 65 / 300
iteration 0 / 20: loss 30.655880
count 66 / 300
iteration 0 / 20: loss 20.766874
count 67 / 300
iteration 0 / 20: loss 22.052172
count 68 / 300
iteration 0 / 20: loss 19.849894
count 69 / 300
iteration 0 / 20: loss 21.862272
count 70 / 300
iteration 0 / 20: loss 24.042186
iteration 0 / 1000: loss 22.101100
iteration 100 / 1000: loss 5.884423
iteration 200 / 1000: loss 4.390483
iteration 300 / 1000: loss 4.954460
iteration 400 / 1000: loss 5.124964
iteration 500 / 1000: loss 4.612712
iteration 600 / 1000: loss 4.728239
iteration 700 / 1000: loss 4.713621
iteration 800 / 1000: loss 5.050876
iteration 900 / 1000: loss 4.798100
Val_accuracy: 0.361, Current best_val: 0.385
count 71 / 300
iteration 0 / 20: loss 31.572124
count 72 / 300
iteration 0 / 20: loss 24.340129
iteration 0 / 1000: loss 23.832673
iteration 100 / 1000: loss 6.437609
iteration 200 / 1000: loss 5.365223
iteration 300 / 1000: loss 6.286175
iteration 400 / 1000: loss 4.379906
iteration 500 / 1000: loss 5.050114
iteration 600 / 1000: loss 6.028924
iteration 700 / 1000: loss 5.468261
iteration 800 / 1000: loss 6.318657
iteration 900 / 1000: loss 5.836085
Val_accuracy: 0.289, Current best_val: 0.385
count 73 / 300
iteration 0 / 20: loss 20.705143
count 74 / 300
iteration 0 / 20: loss 19.477409
iteration 0 / 1000: loss 19.368363
iteration 100 / 1000: loss 5.294448
iteration 200 / 1000: loss 4.979774
iteration 300 / 1000: loss 5.491669
iteration 400 / 1000: loss 5.111605
iteration 500 / 1000: loss 5.145371
iteration 600 / 1000: loss 5.082061
iteration 700 / 1000: loss 4.766225
iteration 800 / 1000: loss 4.862543
iteration 900 / 1000: loss 4.743713
Val_accuracy: 0.379, Current best_val: 0.385
count 75 / 300
iteration 0 / 20: loss 19.955621
count 76 / 300
iteration 0 / 20: loss 23.198313
count 77 / 300
iteration 0 / 20: loss 23.246971
count 78 / 300
iteration 0 / 20: loss 22.734555
iteration 0 / 1000: loss 26.864927
iteration 100 / 1000: loss 8.324150
iteration 200 / 1000: loss 6.833558
iteration 300 / 1000: loss 7.136757
iteration 400 / 1000: loss 6.221461
iteration 500 / 1000: loss 5.935432
iteration 600 / 1000: loss 5.201321
iteration 700 / 1000: loss 6.088373
iteration 800 / 1000: loss 5.424341
iteration 900 / 1000: loss 4.329440
Val_accuracy: 0.348, Current best_val: 0.385
count 79 / 300
iteration 0 / 20: loss 16.500440
iteration 0 / 1000: loss 21.175214
iteration 100 / 1000: loss 6.621194
iteration 200 / 1000: loss 5.339336
iteration 300 / 1000: loss 4.861756
iteration 400 / 1000: loss 5.073304
iteration 500 / 1000: loss 4.967386
iteration 600 / 1000: loss 4.738997
iteration 700 / 1000: loss 4.173003
iteration 800 / 1000: loss 4.916609
iteration 900 / 1000: loss 4.707933
Val_accuracy: 0.372, Current best_val: 0.385
count 80 / 300
iteration 0 / 20: loss 23.267622
count 81 / 300
iteration 0 / 20: loss 20.998430
count 82 / 300
iteration 0 / 20: loss 21.115373
iteration 0 / 1000: loss 22.133817
iteration 100 / 1000: loss 5.337735
iteration 200 / 1000: loss 4.300110
iteration 300 / 1000: loss 5.441841
iteration 400 / 1000: loss 4.355849
iteration 500 / 1000: loss 5.162629
iteration 600 / 1000: loss 4.963214
iteration 700 / 1000: loss 4.465944
iteration 800 / 1000: loss 4.433932
iteration 900 / 1000: loss 4.877654
Val_accuracy: 0.36, Current best_val: 0.385
count 83 / 300
iteration 0 / 20: loss 26.607541
count 84 / 300
iteration 0 / 20: loss 21.137368
count 85 / 300
iteration 0 / 20: loss 21.758313
iteration 0 / 1000: loss 24.049881
iteration 100 / 1000: loss 6.611942
iteration 200 / 1000: loss 8.248458
iteration 300 / 1000: loss 6.768893
iteration 400 / 1000: loss 8.112055
iteration 500 / 1000: loss 7.019736
iteration 600 / 1000: loss 7.215647
iteration 700 / 1000: loss 6.126994
iteration 800 / 1000: loss 8.016675
iteration 900 / 1000: loss 7.762301
Val_accuracy: 0.277, Current best_val: 0.385
count 86 / 300
iteration 0 / 20: loss 18.597776
iteration 0 / 1000: loss 23.947430
iteration 100 / 1000: loss 5.285366
iteration 200 / 1000: loss 5.203686
iteration 300 / 1000: loss 5.072999
iteration 400 / 1000: loss 4.501720
iteration 500 / 1000: loss 4.987670
iteration 600 / 1000: loss 4.852721
iteration 700 / 1000: loss 5.953308
iteration 800 / 1000: loss 4.591105
iteration 900 / 1000: loss 4.807872
Val_accuracy: 0.355, Current best_val: 0.385
count 87 / 300
iteration 0 / 20: loss 18.322090
count 88 / 300
iteration 0 / 20: loss 23.020594
iteration 0 / 1000: loss 18.444594
iteration 100 / 1000: loss 5.921197
iteration 200 / 1000: loss 4.919937
iteration 300 / 1000: loss 5.577291
iteration 400 / 1000: loss 5.268117
iteration 500 / 1000: loss 5.412808
iteration 600 / 1000: loss 5.221651
iteration 700 / 1000: loss 5.399049
iteration 800 / 1000: loss 5.835239
iteration 900 / 1000: loss 5.388931
Val_accuracy: 0.341, Current best_val: 0.385
count 89 / 300
iteration 0 / 20: loss 21.860849
count 90 / 300
iteration 0 / 20: loss 25.747207
count 91 / 300
iteration 0 / 20: loss 23.826987
count 92 / 300
iteration 0 / 20: loss 21.885970
count 93 / 300
iteration 0 / 20: loss 28.696523
count 94 / 300
iteration 0 / 20: loss 22.505017
iteration 0 / 1000: loss 21.914627
iteration 100 / 1000: loss 7.084570
iteration 200 / 1000: loss 5.189869
iteration 300 / 1000: loss 5.060721
iteration 400 / 1000: loss 4.599362
iteration 500 / 1000: loss 4.917093
iteration 600 / 1000: loss 6.061057
iteration 700 / 1000: loss 4.814442
iteration 800 / 1000: loss 5.734337
iteration 900 / 1000: loss 6.143153
Val_accuracy: 0.364, Current best_val: 0.385
count 95 / 300
iteration 0 / 20: loss 20.810730
iteration 0 / 1000: loss 19.650420
iteration 100 / 1000: loss 5.995877
iteration 200 / 1000: loss 7.042431
iteration 300 / 1000: loss 7.647982
iteration 400 / 1000: loss 7.114820
iteration 500 / 1000: loss 5.522841
iteration 600 / 1000: loss 6.640233
iteration 700 / 1000: loss 9.206829
iteration 800 / 1000: loss 6.122477
iteration 900 / 1000: loss 8.268060
Val_accuracy: 0.268, Current best_val: 0.385
count 96 / 300
iteration 0 / 20: loss 20.189324
count 97 / 300
iteration 0 / 20: loss 23.675236
iteration 0 / 1000: loss 19.833702
iteration 100 / 1000: loss 8.846734
iteration 200 / 1000: loss 11.337401
iteration 300 / 1000: loss 9.844127
iteration 400 / 1000: loss 11.011851
iteration 500 / 1000: loss 9.347642
iteration 600 / 1000: loss 7.947673
iteration 700 / 1000: loss 13.452776
iteration 800 / 1000: loss 14.445533
iteration 900 / 1000: loss 8.784544
Val_accuracy: 0.218, Current best_val: 0.385
count 98 / 300
iteration 0 / 20: loss 22.283390
count 99 / 300
iteration 0 / 20: loss 21.658549
count 100 / 300
iteration 0 / 20: loss 24.221705
iteration 0 / 1000: loss 22.063032
iteration 100 / 1000: loss 7.217976
iteration 200 / 1000: loss 6.652122
iteration 300 / 1000: loss 7.128859
iteration 400 / 1000: loss 6.481502
iteration 500 / 1000: loss 9.047135
iteration 600 / 1000: loss 6.219583
iteration 700 / 1000: loss 6.378613
iteration 800 / 1000: loss 5.913993
iteration 900 / 1000: loss 6.837723
Val_accuracy: 0.278, Current best_val: 0.385
count 101 / 300
iteration 0 / 20: loss 22.984618
count 102 / 300
iteration 0 / 20: loss 22.008441
count 103 / 300
iteration 0 / 20: loss 18.433316
iteration 0 / 1000: loss 23.519022
iteration 100 / 1000: loss 8.781718
iteration 200 / 1000: loss 5.119450
iteration 300 / 1000: loss 4.781811
iteration 400 / 1000: loss 5.076969
iteration 500 / 1000: loss 4.814346
iteration 600 / 1000: loss 4.403006
iteration 700 / 1000: loss 4.869851
iteration 800 / 1000: loss 4.593686
iteration 900 / 1000: loss 4.729647
Val_accuracy: 0.373, Current best_val: 0.385
count 104 / 300
iteration 0 / 20: loss 24.179930
iteration 0 / 1000: loss 19.886526
iteration 100 / 1000: loss 4.678151
iteration 200 / 1000: loss 4.751168
iteration 300 / 1000: loss 4.447830
iteration 400 / 1000: loss 4.346740
iteration 500 / 1000: loss 4.977236
iteration 600 / 1000: loss 5.021883
iteration 700 / 1000: loss 5.258936
iteration 800 / 1000: loss 4.197399
iteration 900 / 1000: loss 5.018229
Val_accuracy: 0.365, Current best_val: 0.385
count 105 / 300
iteration 0 / 20: loss 22.251326
iteration 0 / 1000: loss 19.358143
iteration 100 / 1000: loss 6.470184
iteration 200 / 1000: loss 5.358895
iteration 300 / 1000: loss 5.311775
iteration 400 / 1000: loss 5.350971
iteration 500 / 1000: loss 4.823595
iteration 600 / 1000: loss 5.216376
iteration 700 / 1000: loss 5.210660
iteration 800 / 1000: loss 4.806651
iteration 900 / 1000: loss 5.025133
Val_accuracy: 0.358, Current best_val: 0.385
count 106 / 300
```

```
iteration 0 / 20: loss 20.074955
count 107 / 300
iteration 0 / 20: loss 20.780136
count 108 / 300
iteration 0 / 20: loss 23.064239
count 109 / 300
iteration 0 / 20: loss 19.749132
iteration 0 / 1000: loss 21.287928
iteration 100 / 1000: loss 6.291233
iteration 200 / 1000: loss 4.701501
iteration 300 / 1000: loss 5.378386
iteration 400 / 1000: loss 5.042126
iteration 500 / 1000: loss 4.325149
iteration 600 / 1000: loss 4.921302
iteration 700 / 1000: loss 4.714599
iteration 800 / 1000: loss 4.520108
iteration 900 / 1000: loss 5.273143
Val_accuracy: 0.336, Current best_val: 0.385
count 110 / 300
iteration 0 / 20: loss 24.824835
iteration 0 / 1000: loss 21.693140
iteration 100 / 1000: loss 6.181190
iteration 200 / 1000: loss 6.794299
iteration 300 / 1000: loss 9.028502
iteration 400 / 1000: loss 6.084934
iteration 500 / 1000: loss 10.380369
iteration 600 / 1000: loss 9.591354
iteration 700 / 1000: loss 6.605058
iteration 800 / 1000: loss 7.269369
iteration 900 / 1000: loss 7.818759
Val_accuracy: 0.272, Current best_val: 0.385
count 111 / 300
iteration 0 / 20: loss 22.510549
iteration 0 / 1000: loss 28.878488
iteration 100 / 1000: loss 4.993199
iteration 200 / 1000: loss 5.327114
iteration 300 / 1000: loss 5.169291
iteration 400 / 1000: loss 5.510074
iteration 500 / 1000: loss 5.149204
iteration 600 / 1000: loss 5.034979
iteration 700 / 1000: loss 5.433388
iteration 800 / 1000: loss 5.396369
iteration 900 / 1000: loss 4.911558
Val_accuracy: 0.341, Current best_val: 0.385
count 112 / 300
iteration 0 / 20: loss 21.405542
count 113 / 300
iteration 0 / 20: loss 18.955543
```

```
count 114 / 300
iteration 0 / 20: loss 25.090343
count 115 / 300
iteration 0 / 20: loss 22.473180
count 116 / 300
iteration 0 / 20: loss 17.970428
count 117 / 300
iteration 0 / 20: loss 27.348867
count 118 / 300
iteration 0 / 20: loss 21.748715
count 119 / 300
iteration 0 / 20: loss 24.783066
count 120 / 300
iteration 0 / 20: loss 24.672424
count 121 / 300
iteration 0 / 20: loss 21.316681
count 122 / 300
iteration 0 / 20: loss 21.095205
count 123 / 300
iteration 0 / 20: loss 23.635832
count 124 / 300
iteration 0 / 20: loss 20.681214
iteration 0 / 1000: loss 19.959456
iteration 100 / 1000: loss 5.414981
iteration 200 / 1000: loss 4.933941
iteration 300 / 1000: loss 4.755077
iteration 400 / 1000: loss 5.007338
iteration 500 / 1000: loss 5.553840
iteration 600 / 1000: loss 5.568164
iteration 700 / 1000: loss 5.280461
iteration 800 / 1000: loss 5.904795
iteration 900 / 1000: loss 5.431200
Val_accuracy: 0.359, Current best_val: 0.385
count 125 / 300
iteration 0 / 20: loss 21.755758
iteration 0 / 1000: loss 20.197240
iteration 100 / 1000: loss 5.093631
iteration 200 / 1000: loss 7.323967
iteration 300 / 1000: loss 6.262029
iteration 400 / 1000: loss 6.454115
iteration 500 / 1000: loss 5.931523
iteration 600 / 1000: loss 6.645097
iteration 700 / 1000: loss 5.791462
iteration 800 / 1000: loss 6.427238
iteration 900 / 1000: loss 5.264893
Val_accuracy: 0.258, Current best_val: 0.385
count 126 / 300
iteration 0 / 20: loss 21.994906
```

```
count 127 / 300
iteration 0 / 20: loss 23.050749
count 128 / 300
iteration 0 / 20: loss 22.231768
iteration 0 / 1000: loss 18.879706
iteration 100 / 1000: loss 6.755050
iteration 200 / 1000: loss 5.345312
iteration 300 / 1000: loss 5.360784
iteration 400 / 1000: loss 6.833130
iteration 500 / 1000: loss 5.880713
iteration 600 / 1000: loss 4.850082
iteration 700 / 1000: loss 6.104036
iteration 800 / 1000: loss 5.661105
iteration 900 / 1000: loss 5.555720
Val_accuracy: 0.277, Current best_val: 0.385
count 129 / 300
iteration 0 / 20: loss 20.935520
count 130 / 300
iteration 0 / 20: loss 23.332667
iteration 0 / 1000: loss 23.757688
iteration 100 / 1000: loss 8.375345
iteration 200 / 1000: loss 5.817748
iteration 300 / 1000: loss 5.624189
iteration 400 / 1000: loss 4.366546
iteration 500 / 1000: loss 5.405625
iteration 600 / 1000: loss 5.038273
iteration 700 / 1000: loss 5.092012
iteration 800 / 1000: loss 4.178410
iteration 900 / 1000: loss 4.893871
Val_accuracy: 0.385, Current best_val: 0.385
count 131 / 300
iteration 0 / 20: loss 22.232711
count 132 / 300
iteration 0 / 20: loss 21.164579
count 133 / 300
iteration 0 / 20: loss 17.268513
iteration 0 / 1000: loss 22.280645
iteration 100 / 1000: loss 5.893994
iteration 200 / 1000: loss 5.100889
iteration 300 / 1000: loss 4.741489
iteration 400 / 1000: loss 4.459521
iteration 500 / 1000: loss 4.995734
iteration 600 / 1000: loss 4.954202
iteration 700 / 1000: loss 4.733875
iteration 800 / 1000: loss 4.475337
iteration 900 / 1000: loss 4.167568
Val_accuracy: 0.379, Current best_val: 0.385
count 134 / 300
```

```
iteration 0 / 20: loss 19.229107
count 135 / 300
iteration 0 / 20: loss 19.574660
count 136 / 300
iteration 0 / 20: loss 19.551556
count 137 / 300
iteration 0 / 20: loss 31.875354
iteration 0 / 1000: loss 17.580855
iteration 100 / 1000: loss 5.576237
iteration 200 / 1000: loss 5.564536
iteration 300 / 1000: loss 5.384238
iteration 400 / 1000: loss 5.097955
iteration 500 / 1000: loss 5.749792
iteration 600 / 1000: loss 5.476952
iteration 700 / 1000: loss 5.418264
iteration 800 / 1000: loss 5.039185
iteration 900 / 1000: loss 5.731581
Val_accuracy: 0.367, Current best_val: 0.385
count 138 / 300
iteration 0 / 20: loss 21.024865
iteration 0 / 1000: loss 20.257641
iteration 100 / 1000: loss 5.819272
iteration 200 / 1000: loss 4.691542
iteration 300 / 1000: loss 4.791715
iteration 400 / 1000: loss 4.619610
iteration 500 / 1000: loss 4.421143
iteration 600 / 1000: loss 5.248389
iteration 700 / 1000: loss 4.659019
iteration 800 / 1000: loss 5.422939
iteration 900 / 1000: loss 5.277280
Val_accuracy: 0.355, Current best_val: 0.385
count 139 / 300
iteration 0 / 20: loss 22.038066
count 140 / 300
iteration 0 / 20: loss 25.409556
iteration 0 / 1000: loss 23.427698
iteration 100 / 1000: loss 7.833786
iteration 200 / 1000: loss 6.547050
iteration 300 / 1000: loss 8.760698
iteration 400 / 1000: loss 7.578654
iteration 500 / 1000: loss 9.166554
iteration 600 / 1000: loss 7.211494
iteration 700 / 1000: loss 7.513144
iteration 800 / 1000: loss 6.730562
iteration 900 / 1000: loss 8.125756
Val_accuracy: 0.325, Current best_val: 0.385
count 141 / 300
iteration 0 / 20: loss 20.755695
iteration 0 / 1000: loss 22.121548
iteration 100 / 1000: loss 6.188548
iteration 200 / 1000: loss 5.729167
iteration 300 / 1000: loss 6.844271
iteration 400 / 1000: loss 5.100537
iteration 500 / 1000: loss 5.289218
iteration 600 / 1000: loss 5.830996
iteration 700 / 1000: loss 6.291938
iteration 800 / 1000: loss 6.300599
iteration 900 / 1000: loss 5.458662
Val_accuracy: 0.274, Current best_val: 0.385
count 142 / 300
iteration 0 / 20: loss 24.102794
count 143 / 300
iteration 0 / 20: loss 19.033488
count 144 / 300
iteration 0 / 20: loss 26.419545
count 145 / 300
iteration 0 / 20: loss 20.296987
count 146 / 300
iteration 0 / 20: loss 25.560513
count 147 / 300
iteration 0 / 20: loss 19.543761
iteration 0 / 1000: loss 22.115246
iteration 100 / 1000: loss 10.134761
iteration 200 / 1000: loss 12.066236
iteration 300 / 1000: loss 11.309071
iteration 400 / 1000: loss 8.225203
iteration 500 / 1000: loss 8.957563
iteration 600 / 1000: loss 15.053330
iteration 700 / 1000: loss 11.000269
iteration 800 / 1000: loss 10.551859
iteration 900 / 1000: loss 12.451812
Val_accuracy: 0.222, Current best_val: 0.385
count 148 / 300
iteration 0 / 20: loss 32.798083
count 149 / 300
iteration 0 / 20: loss 18.855802
iteration 0 / 1000: loss 21.452210
iteration 100 / 1000: loss 5.132499
iteration 200 / 1000: loss 4.700457
iteration 300 / 1000: loss 4.905685
iteration 400 / 1000: loss 5.519934
iteration 500 / 1000: loss 4.855146
iteration 600 / 1000: loss 5.099475
iteration 700 / 1000: loss 4.940774
iteration 800 / 1000: loss 5.086230
iteration 900 / 1000: loss 5.520567
```

```
Val_accuracy: 0.369, Current best_val: 0.385
count 150 / 300
iteration 0 / 20: loss 21.832070
count 151 / 300
iteration 0 / 20: loss 24.757030
count 152 / 300
iteration 0 / 20: loss 22.415980
count 153 / 300
iteration 0 / 20: loss 20.251604
count 154 / 300
iteration 0 / 20: loss 20.986801
count 155 / 300
iteration 0 / 20: loss 26.490215
iteration 0 / 1000: loss 22.834370
iteration 100 / 1000: loss 7.881342
iteration 200 / 1000: loss 6.706989
iteration 300 / 1000: loss 6.029653
iteration 400 / 1000: loss 5.474786
iteration 500 / 1000: loss 4.717787
iteration 600 / 1000: loss 4.995988
iteration 700 / 1000: loss 5.200779
iteration 800 / 1000: loss 4.448976
iteration 900 / 1000: loss 4.471807
Val_accuracy: 0.378, Current best_val: 0.385
count 156 / 300
iteration 0 / 20: loss 21.843434
count 157 / 300
iteration 0 / 20: loss 19.922293
count 158 / 300
iteration 0 / 20: loss 22.277009
count 159 / 300
iteration 0 / 20: loss 18.882312
count 160 / 300
iteration 0 / 20: loss 23.865294
iteration 0 / 1000: loss 19.826812
iteration 100 / 1000: loss 8.207039
iteration 200 / 1000: loss 8.539918
iteration 300 / 1000: loss 7.430845
iteration 400 / 1000: loss 7.488002
iteration 500 / 1000: loss 5.935840
iteration 600 / 1000: loss 7.926518
iteration 700 / 1000: loss 8.104214
iteration 800 / 1000: loss 8.581565
iteration 900 / 1000: loss 7.547632
Val_accuracy: 0.277, Current best_val: 0.385
count 161 / 300
iteration 0 / 20: loss 21.116393
count 162 / 300
```

```
iteration 0 / 20: loss 23.671528
iteration 0 / 1000: loss 20.510208
iteration 100 / 1000: loss 7.243630
iteration 200 / 1000: loss 5.408677
iteration 300 / 1000: loss 4.869374
iteration 400 / 1000: loss 4.511436
iteration 500 / 1000: loss 4.840280
iteration 600 / 1000: loss 4.551846
iteration 700 / 1000: loss 4.390867
iteration 800 / 1000: loss 4.261822
iteration 900 / 1000: loss 4.321731
Val_accuracy: 0.359, Current best_val: 0.385
count 163 / 300
iteration 0 / 20: loss 24.046945
count 164 / 300
iteration 0 / 20: loss 23.688437
count 165 / 300
iteration 0 / 20: loss 23.034241
count 166 / 300
iteration 0 / 20: loss 23.532735
count 167 / 300
iteration 0 / 20: loss 22.695125
count 168 / 300
iteration 0 / 20: loss 27.145264
iteration 0 / 1000: loss 22.096568
iteration 100 / 1000: loss 6.384905
iteration 200 / 1000: loss 6.904568
iteration 300 / 1000: loss 5.198416
iteration 400 / 1000: loss 5.308381
iteration 500 / 1000: loss 5.919747
iteration 600 / 1000: loss 5.944620
iteration 700 / 1000: loss 5.427763
iteration 800 / 1000: loss 6.644075
iteration 900 / 1000: loss 5.817461
Val_accuracy: 0.313, Current best_val: 0.385
count 169 / 300
iteration 0 / 20: loss 21.299300
count 170 / 300
iteration 0 / 20: loss 24.410887
count 171 / 300
iteration 0 / 20: loss 21.307598
iteration 0 / 1000: loss 18.743073
iteration 100 / 1000: loss 6.403331
iteration 200 / 1000: loss 4.400023
iteration 300 / 1000: loss 4.580350
iteration 400 / 1000: loss 4.937781
iteration 500 / 1000: loss 4.466740
iteration 600 / 1000: loss 4.915757
iteration 700 / 1000: loss 5.176374
iteration 800 / 1000: loss 4.438565
iteration 900 / 1000: loss 4.165605
Val_accuracy: 0.359, Current best_val: 0.385
count 172 / 300
iteration 0 / 20: loss 20.486337
count 173 / 300
iteration 0 / 20: loss 20.100777
iteration 0 / 1000: loss 21.850492
iteration 100 / 1000: loss 7.193977
iteration 200 / 1000: loss 5.706966
iteration 300 / 1000: loss 4.854799
iteration 400 / 1000: loss 5.032539
iteration 500 / 1000: loss 4.488563
iteration 600 / 1000: loss 4.531713
iteration 700 / 1000: loss 4.901376
iteration 800 / 1000: loss 5.166730
iteration 900 / 1000: loss 4.495969
Val_accuracy: 0.379, Current best_val: 0.385
count 174 / 300
iteration 0 / 20: loss 27.227081
count 175 / 300
iteration 0 / 20: loss 26.825441
iteration 0 / 1000: loss 20.343193
iteration 100 / 1000: loss 5.890170
iteration 200 / 1000: loss 5.081761
iteration 300 / 1000: loss 5.222661
iteration 400 / 1000: loss 4.727905
iteration 500 / 1000: loss 4.774549
iteration 600 / 1000: loss 5.554491
iteration 700 / 1000: loss 5.027468
iteration 800 / 1000: loss 5.457218
iteration 900 / 1000: loss 5.317625
Val_accuracy: 0.356, Current best_val: 0.385
count 176 / 300
iteration 0 / 20: loss 20.237968
count 177 / 300
iteration 0 / 20: loss 20.082474
count 178 / 300
iteration 0 / 20: loss 20.921159
count 179 / 300
iteration 0 / 20: loss 22.156070
count 180 / 300
iteration 0 / 20: loss 21.644092
count 181 / 300
iteration 0 / 20: loss 22.920997
count 182 / 300
iteration 0 / 20: loss 20.466953
```

```
count 183 / 300
iteration 0 / 20: loss 20.144900
count 184 / 300
iteration 0 / 20: loss 18.534640
iteration 0 / 1000: loss 21.309954
iteration 100 / 1000: loss 5.226604
iteration 200 / 1000: loss 4.778565
iteration 300 / 1000: loss 5.015466
iteration 400 / 1000: loss 5.001683
iteration 500 / 1000: loss 6.050476
iteration 600 / 1000: loss 5.004575
iteration 700 / 1000: loss 5.511686
iteration 800 / 1000: loss 5.212727
iteration 900 / 1000: loss 5.064357
Val_accuracy: 0.335, Current best_val: 0.385
count 185 / 300
iteration 0 / 20: loss 21.076511
iteration 0 / 1000: loss 18.010832
iteration 100 / 1000: loss 16.477917
iteration 200 / 1000: loss 13.291436
iteration 300 / 1000: loss 8.734024
iteration 400 / 1000: loss 10.946303
iteration 500 / 1000: loss 11.858523
iteration 600 / 1000: loss 10.061494
iteration 700 / 1000: loss 9.326109
iteration 800 / 1000: loss 7.399237
iteration 900 / 1000: loss 8.596526
Val_accuracy: 0.245, Current best_val: 0.385
count 186 / 300
iteration 0 / 20: loss 19.759890
iteration 0 / 1000: loss 24.872213
iteration 100 / 1000: loss 9.343536
iteration 200 / 1000: loss 6.752544
iteration 300 / 1000: loss 7.759312
iteration 400 / 1000: loss 9.078988
iteration 500 / 1000: loss 7.290790
iteration 600 / 1000: loss 8.067185
iteration 700 / 1000: loss 8.535159
iteration 800 / 1000: loss 10.140541
iteration 900 / 1000: loss 8.178457
Val_accuracy: 0.295, Current best_val: 0.385
count 187 / 300
iteration 0 / 20: loss 19.864661
count 188 / 300
iteration 0 / 20: loss 23.681111
count 189 / 300
iteration 0 / 20: loss 21.029017
iteration 0 / 1000: loss 20.203215
iteration 100 / 1000: loss 5.316578
iteration 200 / 1000: loss 5.115103
iteration 300 / 1000: loss 5.071208
iteration 400 / 1000: loss 4.444466
iteration 500 / 1000: loss 4.853926
iteration 600 / 1000: loss 4.921308
iteration 700 / 1000: loss 4.734009
iteration 800 / 1000: loss 5.435730
iteration 900 / 1000: loss 5.172671
Val_accuracy: 0.373, Current best_val: 0.385
count 190 / 300
iteration 0 / 20: loss 20.790215
count 191 / 300
iteration 0 / 20: loss 27.169017
iteration 0 / 1000: loss 23.369140
iteration 100 / 1000: loss 10.231082
iteration 200 / 1000: loss 12.355639
iteration 300 / 1000: loss 6.741373
iteration 400 / 1000: loss 9.220337
iteration 500 / 1000: loss 11.981693
iteration 600 / 1000: loss 10.967336
iteration 700 / 1000: loss 11.027479
iteration 800 / 1000: loss 11.007030
iteration 900 / 1000: loss 10.445694
Val_accuracy: 0.275, Current best_val: 0.385
count 192 / 300
iteration 0 / 20: loss 23.033163
count 193 / 300
iteration 0 / 20: loss 21.141919
count 194 / 300
iteration 0 / 20: loss 23.237477
iteration 0 / 1000: loss 24.593631
iteration 100 / 1000: loss 9.918783
iteration 200 / 1000: loss 10.648284
iteration 300 / 1000: loss 11.936831
iteration 400 / 1000: loss 11.234207
iteration 500 / 1000: loss 11.654835
iteration 600 / 1000: loss 11.080591
iteration 700 / 1000: loss 9.836947
iteration 800 / 1000: loss 9.628206
iteration 900 / 1000: loss 12.721885
Val_accuracy: 0.303, Current best_val: 0.385
count 195 / 300
iteration 0 / 20: loss 18.590736
count 196 / 300
iteration 0 / 20: loss 22.728046
count 197 / 300
iteration 0 / 20: loss 21.300023
iteration 0 / 1000: loss 18.370869
iteration 100 / 1000: loss 6.006737
iteration 200 / 1000: loss 6.156321
iteration 300 / 1000: loss 5.998627
iteration 400 / 1000: loss 5.395290
iteration 500 / 1000: loss 7.569271
iteration 600 / 1000: loss 6.532710
iteration 700 / 1000: loss 5.811210
iteration 800 / 1000: loss 6.717094
iteration 900 / 1000: loss 5.900104
Val_accuracy: 0.322, Current best_val: 0.385
count 198 / 300
iteration 0 / 20: loss 29.824031
iteration 0 / 1000: loss 23.927567
iteration 100 / 1000: loss 7.097162
iteration 200 / 1000: loss 5.222865
iteration 300 / 1000: loss 5.036493
iteration 400 / 1000: loss 4.974559
iteration 500 / 1000: loss 5.276946
iteration 600 / 1000: loss 4.493809
iteration 700 / 1000: loss 4.405790
iteration 800 / 1000: loss 4.613328
iteration 900 / 1000: loss 5.100649
Val_accuracy: 0.379, Current best_val: 0.385
count 199 / 300
iteration 0 / 20: loss 25.145587
iteration 0 / 1000: loss 23.973144
iteration 100 / 1000: loss 9.436779
iteration 200 / 1000: loss 7.727483
iteration 300 / 1000: loss 8.134882
iteration 400 / 1000: loss 9.079554
iteration 500 / 1000: loss 8.317271
iteration 600 / 1000: loss 7.947806
iteration 700 / 1000: loss 8.633757
iteration 800 / 1000: loss 8.037590
iteration 900 / 1000: loss 8.443732
Val_accuracy: 0.266, Current best_val: 0.385
count 200 / 300
iteration 0 / 20: loss 25.139483
iteration 0 / 1000: loss 22.210575
iteration 100 / 1000: loss 8.524981
iteration 200 / 1000: loss 7.062114
iteration 300 / 1000: loss 11.600729
iteration 400 / 1000: loss 8.891720
iteration 500 / 1000: loss 9.122832
iteration 600 / 1000: loss 9.472207
iteration 700 / 1000: loss 8.651052
iteration 800 / 1000: loss 13.072931
iteration 900 / 1000: loss 7.100864
Val_accuracy: 0.226, Current best_val: 0.385
count 201 / 300
iteration 0 / 20: loss 22.434728
count 202 / 300
iteration 0 / 20: loss 17.775243
count 203 / 300
iteration 0 / 20: loss 23.011791
count 204 / 300
iteration 0 / 20: loss 22.139309
iteration 0 / 1000: loss 23.400729
iteration 100 / 1000: loss 4.701570
iteration 200 / 1000: loss 4.792659
iteration 300 / 1000: loss 5.625295
iteration 400 / 1000: loss 5.148322
iteration 500 / 1000: loss 5.067673
iteration 600 / 1000: loss 5.163472
iteration 700 / 1000: loss 5.770937
iteration 800 / 1000: loss 4.804334
iteration 900 / 1000: loss 5.900002
Val_accuracy: 0.358, Current best_val: 0.385
count 205 / 300
iteration 0 / 20: loss 17.758985
count 206 / 300
iteration 0 / 20: loss 23.619878
count 207 / 300
iteration 0 / 20: loss 22.318238
count 208 / 300
iteration 0 / 20: loss 24.623609
iteration 0 / 1000: loss 20.246016
iteration 100 / 1000: loss 10.470357
iteration 200 / 1000: loss 17.952705
iteration 300 / 1000: loss 21.823255
iteration 400 / 1000: loss 11.713590
iteration 500 / 1000: loss 18.262444
iteration 600 / 1000: loss 11.497327
iteration 700 / 1000: loss 19.857953
iteration 800 / 1000: loss 15.203306
iteration 900 / 1000: loss 15.882233
Val_accuracy: 0.213, Current best_val: 0.385
count 209 / 300
iteration 0 / 20: loss 22.724644
iteration 0 / 1000: loss 21.186908
iteration 100 / 1000: loss 6.151610
iteration 200 / 1000: loss 8.711356
iteration 300 / 1000: loss 6.594517
iteration 400 / 1000: loss 7.178845
iteration 500 / 1000: loss 7.155324
iteration 600 / 1000: loss 6.499782
iteration 700 / 1000: loss 7.112271
iteration 800 / 1000: loss 6.781053
iteration 900 / 1000: loss 8.997410
Val_accuracy: 0.264, Current best_val: 0.385
count 210 / 300
iteration 0 / 20: loss 27.142936
count 211 / 300
iteration 0 / 20: loss 25.708571
count 212 / 300
iteration 0 / 20: loss 18.286391
count 213 / 300
iteration 0 / 20: loss 22.622903
iteration 0 / 1000: loss 22.791978
iteration 100 / 1000: loss 5.108765
iteration 200 / 1000: loss 5.173664
iteration 300 / 1000: loss 5.104513
iteration 400 / 1000: loss 4.609037
iteration 500 / 1000: loss 4.553192
iteration 600 / 1000: loss 5.272035
iteration 700 / 1000: loss 4.501923
iteration 800 / 1000: loss 4.483049
iteration 900 / 1000: loss 4.834269
Val_accuracy: 0.371, Current best_val: 0.385
count 214 / 300
iteration 0 / 20: loss 24.840613
count 215 / 300
iteration 0 / 20: loss 23.307411
iteration 0 / 1000: loss 18.931032
iteration 100 / 1000: loss 7.492429
iteration 200 / 1000: loss 5.863090
iteration 300 / 1000: loss 4.685194
iteration 400 / 1000: loss 4.365987
iteration 500 / 1000: loss 4.461007
iteration 600 / 1000: loss 4.595930
iteration 700 / 1000: loss 4.851336
iteration 800 / 1000: loss 4.704618
iteration 900 / 1000: loss 5.036116
Val_accuracy: 0.381, Current best_val: 0.385
count 216 / 300
iteration 0 / 20: loss 19.596718
count 217 / 300
iteration 0 / 20: loss 17.525684
count 218 / 300
iteration 0 / 20: loss 20.792122
count 219 / 300
iteration 0 / 20: loss 17.540012
count 220 / 300
```

```
iteration 0 / 20: loss 19.021590
iteration 0 / 1000: loss 23.697371
iteration 100 / 1000: loss 12.616559
iteration 200 / 1000: loss 11.368354
iteration 300 / 1000: loss 10.915189
iteration 400 / 1000: loss 9.131154
iteration 500 / 1000: loss 9.639756
iteration 600 / 1000: loss 11.703134
iteration 700 / 1000: loss 9.893736
iteration 800 / 1000: loss 12.451833
iteration 900 / 1000: loss 11.599818
Val_accuracy: 0.249, Current best_val: 0.385
count 221 / 300
iteration 0 / 20: loss 23.457849
iteration 0 / 1000: loss 20.717007
iteration 100 / 1000: loss 8.405415
iteration 200 / 1000: loss 6.669079
iteration 300 / 1000: loss 6.649604
iteration 400 / 1000: loss 5.073213
iteration 500 / 1000: loss 5.350807
iteration 600 / 1000: loss 5.027510
iteration 700 / 1000: loss 4.725580
iteration 800 / 1000: loss 4.099500
iteration 900 / 1000: loss 3.707056
best_val updated to 0.396000
count 222 / 300
iteration 0 / 20: loss 22.780439
count 223 / 300
iteration 0 / 20: loss 21.198284
count 224 / 300
iteration 0 / 20: loss 18.791021
iteration 0 / 1000: loss 21.424032
iteration 100 / 1000: loss 6.629220
iteration 200 / 1000: loss 5.780380
iteration 300 / 1000: loss 5.965054
iteration 400 / 1000: loss 7.489219
iteration 500 / 1000: loss 6.100768
iteration 600 / 1000: loss 5.829290
iteration 700 / 1000: loss 6.219927
iteration 800 / 1000: loss 5.766776
iteration 900 / 1000: loss 5.511809
Val_accuracy: 0.274, Current best_val: 0.396
count 225 / 300
iteration 0 / 20: loss 18.515806
count 226 / 300
iteration 0 / 20: loss 25.418300
count 227 / 300
iteration 0 / 20: loss 23.611199
iteration 0 / 1000: loss 23.947151
iteration 100 / 1000: loss 6.615594
iteration 200 / 1000: loss 4.795168
iteration 300 / 1000: loss 4.420452
iteration 400 / 1000: loss 4.307619
iteration 500 / 1000: loss 4.441803
iteration 600 / 1000: loss 4.490578
iteration 700 / 1000: loss 4.867015
iteration 800 / 1000: loss 5.196335
iteration 900 / 1000: loss 4.880715
Val_accuracy: 0.388, Current best_val: 0.396
count 228 / 300
iteration 0 / 20: loss 29.624028
count 229 / 300
iteration 0 / 20: loss 24.294982
iteration 0 / 1000: loss 28.296039
iteration 100 / 1000: loss 6.199235
iteration 200 / 1000: loss 6.270550
iteration 300 / 1000: loss 5.557313
iteration 400 / 1000: loss 5.871995
iteration 500 / 1000: loss 5.652646
iteration 600 / 1000: loss 5.578753
iteration 700 / 1000: loss 5.042440
iteration 800 / 1000: loss 6.569593
iteration 900 / 1000: loss 5.536351
Val_accuracy: 0.32, Current best_val: 0.396
count 230 / 300
iteration 0 / 20: loss 18.786250
iteration 0 / 1000: loss 24.239553
iteration 100 / 1000: loss 6.750363
iteration 200 / 1000: loss 5.526301
iteration 300 / 1000: loss 5.739857
iteration 400 / 1000: loss 4.781191
iteration 500 / 1000: loss 5.060641
iteration 600 / 1000: loss 4.705865
iteration 700 / 1000: loss 4.919996
iteration 800 / 1000: loss 5.779591
iteration 900 / 1000: loss 4.474746
Val_accuracy: 0.36, Current best_val: 0.396
count 231 / 300
iteration 0 / 20: loss 23.437518
iteration 0 / 1000: loss 24.037936
iteration 100 / 1000: loss 5.461482
iteration 200 / 1000: loss 5.583890
iteration 300 / 1000: loss 4.951198
iteration 400 / 1000: loss 5.115520
iteration 500 / 1000: loss 5.510333
iteration 600 / 1000: loss 5.033044
iteration 700 / 1000: loss 4.431489
iteration 800 / 1000: loss 4.897523
iteration 900 / 1000: loss 4.540934
Val_accuracy: 0.361, Current best_val: 0.396
count 232 / 300
iteration 0 / 20: loss 23.038473
iteration 0 / 1000: loss 25.531438
iteration 100 / 1000: loss 5.791687
iteration 200 / 1000: loss 5.711986
iteration 300 / 1000: loss 5.822935
iteration 400 / 1000: loss 5.854430
iteration 500 / 1000: loss 5.844725
iteration 600 / 1000: loss 5.907744
iteration 700 / 1000: loss 5.584340
iteration 800 / 1000: loss 5.723703
iteration 900 / 1000: loss 5.833867
Val_accuracy: 0.29, Current best_val: 0.396
count 233 / 300
iteration 0 / 20: loss 20.898673
count 234 / 300
iteration 0 / 20: loss 25.620235
count 235 / 300
iteration 0 / 20: loss 22.412920
iteration 0 / 1000: loss 20.733685
iteration 100 / 1000: loss 12.034208
iteration 200 / 1000: loss 12.895552
iteration 300 / 1000: loss 13.026561
iteration 400 / 1000: loss 13.129289
iteration 500 / 1000: loss 10.961343
iteration 600 / 1000: loss 11.481070
iteration 700 / 1000: loss 13.411563
iteration 800 / 1000: loss 15.037926
iteration 900 / 1000: loss 10.242298
Val_accuracy: 0.206, Current best_val: 0.396
count 236 / 300
iteration 0 / 20: loss 17.166150
count 237 / 300
iteration 0 / 20: loss 19.760830
iteration 0 / 1000: loss 24.645376
iteration 100 / 1000: loss 5.983040
iteration 200 / 1000: loss 6.476305
iteration 300 / 1000: loss 5.673967
iteration 400 / 1000: loss 5.285167
iteration 500 / 1000: loss 6.283310
iteration 600 / 1000: loss 5.841700
iteration 700 / 1000: loss 5.835868
iteration 800 / 1000: loss 5.140215
iteration 900 / 1000: loss 5.272904
```

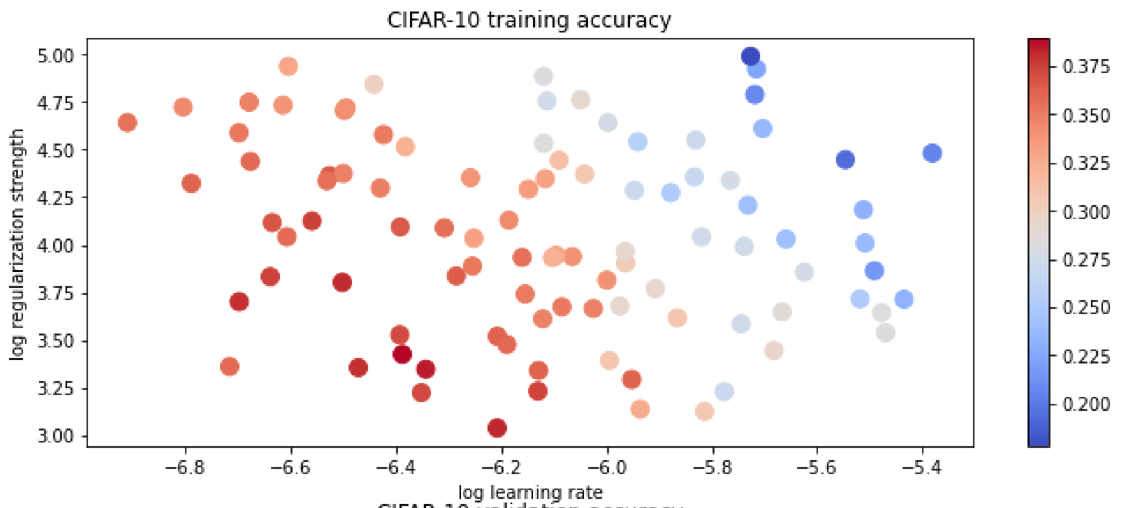
```
Val_accuracy: 0.295, Current best_val: 0.396
count 238 / 300
iteration 0 / 20: loss 19.865541
count 239 / 300
iteration 0 / 20: loss 19.587278
count 240 / 300
iteration 0 / 20: loss 24.058288
count 241 / 300
iteration 0 / 20: loss 24.964051
count 242 / 300
iteration 0 / 20: loss 19.320488
count 243 / 300
iteration 0 / 20: loss 19.167913
count 244 / 300
iteration 0 / 20: loss 21.701130
count 245 / 300
iteration 0 / 20: loss 17.740518
count 246 / 300
iteration 0 / 20: loss 21.168512
iteration 0 / 1000: loss 23.257539
iteration 100 / 1000: loss 5.251701
iteration 200 / 1000: loss 5.552848
iteration 300 / 1000: loss 5.619879
iteration 400 / 1000: loss 6.048932
iteration 500 / 1000: loss 6.151151
iteration 600 / 1000: loss 5.537416
iteration 700 / 1000: loss 5.546057
iteration 800 / 1000: loss 5.847054
iteration 900 / 1000: loss 5.160415
Val_accuracy: 0.3, Current best_val: 0.396
count 247 / 300
iteration 0 / 20: loss 21.040896
count 248 / 300
iteration 0 / 20: loss 24.469429
count 249 / 300
iteration 0 / 20: loss 23.933953
count 250 / 300
iteration 0 / 20: loss 24.410744
count 251 / 300
iteration 0 / 20: loss 16.750282
count 252 / 300
iteration 0 / 20: loss 21.156491
count 253 / 300
iteration 0 / 20: loss 19.905597
count 254 / 300
iteration 0 / 20: loss 20.492724
iteration 0 / 1000: loss 20.973602
iteration 100 / 1000: loss 5.375589
iteration 200 / 1000: loss 6.199199
iteration 300 / 1000: loss 5.406359
iteration 400 / 1000: loss 5.640333
iteration 500 / 1000: loss 5.737611
iteration 600 / 1000: loss 5.206541
iteration 700 / 1000: loss 5.413189
iteration 800 / 1000: loss 5.064050
iteration 900 / 1000: loss 5.615422
Val_accuracy: 0.334, Current best_val: 0.396
count 255 / 300
iteration 0 / 20: loss 21.301079
count 256 / 300
iteration 0 / 20: loss 22.870020
count 257 / 300
iteration 0 / 20: loss 24.587387
count 258 / 300
iteration 0 / 20: loss 15.908832
iteration 0 / 1000: loss 19.245562
iteration 100 / 1000: loss 5.180803
iteration 200 / 1000: loss 4.971520
iteration 300 / 1000: loss 5.154735
iteration 400 / 1000: loss 4.839011
iteration 500 / 1000: loss 4.790252
iteration 600 / 1000: loss 5.782680
iteration 700 / 1000: loss 5.181137
iteration 800 / 1000: loss 4.635063
iteration 900 / 1000: loss 5.079504
Val_accuracy: 0.371, Current best_val: 0.396
count 259 / 300
iteration 0 / 20: loss 30.386142
iteration 0 / 1000: loss 18.795292
iteration 100 / 1000: loss 5.539616
iteration 200 / 1000: loss 5.241250
iteration 300 / 1000: loss 4.776462
iteration 400 / 1000: loss 4.548003
iteration 500 / 1000: loss 5.273172
iteration 600 / 1000: loss 5.219242
iteration 700 / 1000: loss 5.259595
iteration 800 / 1000: loss 4.963770
iteration 900 / 1000: loss 4.937058
Val_accuracy: 0.337, Current best_val: 0.396
count 260 / 300
iteration 0 / 20: loss 22.474280
count 261 / 300
iteration 0 / 20: loss 29.295354
count 262 / 300
iteration 0 / 20: loss 23.007043
iteration 0 / 1000: loss 27.012997
iteration 100 / 1000: loss 5.362405
iteration 200 / 1000: loss 4.920352
iteration 300 / 1000: loss 5.385749
iteration 400 / 1000: loss 5.541626
iteration 500 / 1000: loss 5.291113
iteration 600 / 1000: loss 4.938869
iteration 700 / 1000: loss 4.891785
iteration 800 / 1000: loss 5.194984
iteration 900 / 1000: loss 4.609555
Val_accuracy: 0.334, Current best_val: 0.396
count 263 / 300
iteration 0 / 20: loss 29.253177
count 264 / 300
iteration 0 / 20: loss 20.574927
iteration 0 / 1000: loss 21.326516
iteration 100 / 1000: loss 7.210555
iteration 200 / 1000: loss 5.776568
iteration 300 / 1000: loss 6.525099
iteration 400 / 1000: loss 6.640140
iteration 500 / 1000: loss 6.919439
iteration 600 / 1000: loss 6.030147
iteration 700 / 1000: loss 6.299959
iteration 800 / 1000: loss 4.945766
iteration 900 / 1000: loss 6.388706
Val_accuracy: 0.268, Current best_val: 0.396
count 265 / 300
iteration 0 / 20: loss 21.860921
count 266 / 300
iteration 0 / 20: loss 23.201115
count 267 / 300
iteration 0 / 20: loss 18.180193
count 268 / 300
iteration 0 / 20: loss 20.170983
count 269 / 300
iteration 0 / 20: loss 25.462070
iteration 0 / 1000: loss 18.021024
iteration 100 / 1000: loss 12.027310
iteration 200 / 1000: loss 8.411468
iteration 300 / 1000: loss 10.627069
iteration 400 / 1000: loss 11.967976
iteration 500 / 1000: loss 9.190405
iteration 600 / 1000: loss 12.094446
iteration 700 / 1000: loss 8.791646
iteration 800 / 1000: loss 8.376407
iteration 900 / 1000: loss 13.378803
Val_accuracy: 0.218, Current best_val: 0.396
count 270 / 300
iteration 0 / 20: loss 24.264312
iteration 0 / 1000: loss 22.447006
iteration 100 / 1000: loss 5.775852
iteration 200 / 1000: loss 5.705880
iteration 300 / 1000: loss 4.284009
iteration 400 / 1000: loss 5.900461
iteration 500 / 1000: loss 5.218527
iteration 600 / 1000: loss 5.148556
iteration 700 / 1000: loss 5.571825
iteration 800 / 1000: loss 5.513252
iteration 900 / 1000: loss 4.608261
Val_accuracy: 0.364, Current best_val: 0.396
count 271 / 300
iteration 0 / 20: loss 21.848542
count 272 / 300
iteration 0 / 20: loss 20.184315
count 273 / 300
iteration 0 / 20: loss 19.179619
iteration 0 / 1000: loss 23.230691
iteration 100 / 1000: loss 6.087259
iteration 200 / 1000: loss 5.113012
iteration 300 / 1000: loss 5.587806
iteration 400 / 1000: loss 5.547817
iteration 500 / 1000: loss 5.527060
iteration 600 / 1000: loss 5.911353
iteration 700 / 1000: loss 5.364860
iteration 800 / 1000: loss 5.367343
iteration 900 / 1000: loss 5.019608
Val_accuracy: 0.328, Current best_val: 0.396
count 274 / 300
iteration 0 / 20: loss 24.354014
count 275 / 300
iteration 0 / 20: loss 23.580808
count 276 / 300
iteration 0 / 20: loss 17.156373
count 277 / 300
iteration 0 / 20: loss 23.058901
count 278 / 300
iteration 0 / 20: loss 19.139091
count 279 / 300
iteration 0 / 20: loss 21.636394
count 280 / 300
iteration 0 / 20: loss 24.372505
count 281 / 300
iteration 0 / 20: loss 20.598055
iteration 0 / 1000: loss 26.376188
iteration 100 / 1000: loss 6.784001
iteration 200 / 1000: loss 4.878020
iteration 300 / 1000: loss 4.812641
iteration 400 / 1000: loss 5.545743
iteration 500 / 1000: loss 4.351305
iteration 600 / 1000: loss 4.395209
iteration 700 / 1000: loss 4.846845
iteration 800 / 1000: loss 4.754233
iteration 900 / 1000: loss 4.294288
Val_accuracy: 0.37, Current best_val: 0.396
count 282 / 300
iteration 0 / 20: loss 20.498066
iteration 0 / 1000: loss 17.828765
iteration 100 / 1000: loss 5.175265
iteration 200 / 1000: loss 5.315121
iteration 300 / 1000: loss 5.006785
iteration 400 / 1000: loss 5.038353
iteration 500 / 1000: loss 5.651754
iteration 600 / 1000: loss 5.124935
iteration 700 / 1000: loss 5.388971
iteration 800 / 1000: loss 5.114094
iteration 900 / 1000: loss 5.433668
Val_accuracy: 0.344, Current best_val: 0.396
count 283 / 300
iteration 0 / 20: loss 16.761569
iteration 0 / 1000: loss 21.541717
iteration 100 / 1000: loss 5.658408
iteration 200 / 1000: loss 4.899425
iteration 300 / 1000: loss 4.966057
iteration 400 / 1000: loss 5.313872
iteration 500 / 1000: loss 4.950384
iteration 600 / 1000: loss 4.609170
iteration 700 / 1000: loss 5.134254
iteration 800 / 1000: loss 5.174341
iteration 900 / 1000: loss 5.489843
Val_accuracy: 0.342, Current best_val: 0.396
count 284 / 300
iteration 0 / 20: loss 22.021890
iteration 0 / 1000: loss 32.152428
iteration 100 / 1000: loss 5.120480
iteration 200 / 1000: loss 6.065857
iteration 300 / 1000: loss 6.380973
iteration 400 / 1000: loss 5.430526
iteration 500 / 1000: loss 5.658071
iteration 600 / 1000: loss 5.745133
iteration 700 / 1000: loss 6.330432
iteration 800 / 1000: loss 5.018753
iteration 900 / 1000: loss 4.991784
Val_accuracy: 0.343, Current best_val: 0.396
count 285 / 300
iteration 0 / 20: loss 21.195136
```

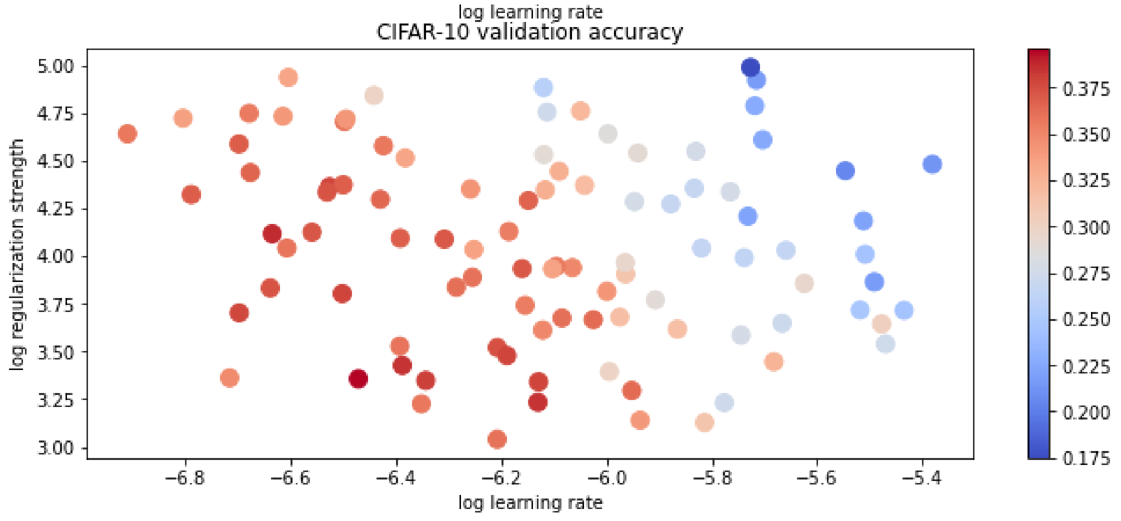
```
count 286 / 300
iteration 0 / 20: loss 22.188951
count 287 / 300
iteration 0 / 20: loss 21.753645
count 288 / 300
iteration 0 / 20: loss 23.010250
count 289 / 300
iteration 0 / 20: loss 24.785902
count 290 / 300
iteration 0 / 20: loss 22.015704
iteration 0 / 1000: loss 21.115196
iteration 100 / 1000: loss 7.076041
iteration 200 / 1000: loss 5.879344
iteration 300 / 1000: loss 6.024624
iteration 400 / 1000: loss 5.821341
iteration 500 / 1000: loss 5.351220
iteration 600 / 1000: loss 5.360119
iteration 700 / 1000: loss 5.543679
iteration 800 / 1000: loss 5.484511
iteration 900 / 1000: loss 5.141203
Val_accuracy: 0.317, Current best_val: 0.396
count 291 / 300
iteration 0 / 20: loss 21.600718
count 292 / 300
iteration 0 / 20: loss 20.832639
iteration 0 / 1000: loss 20.591991
iteration 100 / 1000: loss 9.771671
iteration 200 / 1000: loss 12.366101
iteration 300 / 1000: loss 9.333157
iteration 400 / 1000: loss 10.675666
iteration 500 / 1000: loss 12.031693
iteration 600 / 1000: loss 9.865852
iteration 700 / 1000: loss 11.966375
iteration 800 / 1000: loss 11.462652
iteration 900 / 1000: loss 11.194786
Val_accuracy: 0.175, Current best_val: 0.396
count 293 / 300
iteration 0 / 20: loss 22.749538
iteration 0 / 1000: loss 20.948077
iteration 100 / 1000: loss 8.276485
iteration 200 / 1000: loss 7.820649
iteration 300 / 1000: loss 8.847044
iteration 400 / 1000: loss 8.501508
iteration 500 / 1000: loss 8.937636
iteration 600 / 1000: loss 10.504548
iteration 700 / 1000: loss 10.329468
iteration 800 / 1000: loss 9.389353
iteration 900 / 1000: loss 7.555502
```

```
Val_accuracy: 0.227, Current best_val: 0.396
count 294 / 300
iteration 0 / 20: loss 26.092780
count 295 / 300
iteration 0 / 20: loss 22.177297
count 296 / 300
iteration 0 / 20: loss 24.197940
iteration 0 / 1000: loss 21.866979
iteration 100 / 1000: loss 6.309496
iteration 200 / 1000: loss 4.708148
iteration 300 / 1000: loss 4.879110
iteration 400 / 1000: loss 5.937070
iteration 500 / 1000: loss 4.616215
iteration 600 / 1000: loss 5.154486
iteration 700 / 1000: loss 4.636147
iteration 800 / 1000: loss 5.665015
iteration 900 / 1000: loss 4.777165
Val_accuracy: 0.3, Current best_val: 0.396
count 297 / 300
iteration 0 / 20: loss 21.173826
count 298 / 300
iteration 0 / 20: loss 22.329611
iteration 0 / 1000: loss 23.016170
iteration 100 / 1000: loss 6.484549
iteration 200 / 1000: loss 4.467600
iteration 300 / 1000: loss 4.102351
iteration 400 / 1000: loss 5.189265
iteration 500 / 1000: loss 4.725908
iteration 600 / 1000: loss 4.560833
iteration 700 / 1000: loss 5.732345
iteration 800 / 1000: loss 4.846184
iteration 900 / 1000: loss 4.185134
Val_accuracy: 0.35, Current best_val: 0.396
count 299 / 300
iteration 0 / 20: loss 25.007884
lr 1.230550e-07 reg 4.369127e+04 train accuracy: 0.355755 val accuracy: 0.358000
lr 1.569799e-07 reg 5.272361e+04 train accuracy: 0.344367 val accuracy: 0.337000
lr 1.626796e-07 reg 2.098480e+04 train accuracy: 0.361592 val accuracy: 0.370000
lr 1.924467e-07 reg 2.297065e+03 train accuracy: 0.358571 val accuracy: 0.348000
lr 2.003894e-07 reg 3.865475e+04 train accuracy: 0.353082 val accuracy: 0.370000
lr 2.005806e-07 reg 5.030609e+03 train accuracy: 0.380694 val accuracy: 0.378000
lr 2.094690e-07 reg 5.595308e+04 train accuracy: 0.348918 val accuracy: 0.356000
lr 2.106086e-07 reg 2.734659e+04 train accuracy: 0.359531 val accuracy: 0.366000
lr 2.298031e-07 reg 6.794671e+03 train accuracy: 0.374510 val accuracy: 0.373000
lr 2.318124e-07 reg 1.306915e+04 train accuracy: 0.366633 val accuracy: 0.388000
lr 2.429416e-07 reg 5.399996e+04 train accuracy: 0.340061 val accuracy: 0.341000
lr 2.470444e-07 reg 1.101336e+04 train accuracy: 0.358694 val accuracy: 0.359000
lr 2.487243e-07 reg 8.624477e+04 train accuracy: 0.329816 val accuracy: 0.334000
lr 2.757099e-07 reg 1.332200e+04 train accuracy: 0.374939 val accuracy: 0.372000
lr 2.945170e-07 reg 2.162446e+04 train accuracy: 0.356571 val accuracy: 0.373000
lr 2.978122e-07 reg 2.303323e+04 train accuracy: 0.365571 val accuracy: 0.379000
lr 3.148807e-07 reg 6.352960e+03 train accuracy: 0.381633 val accuracy: 0.379000
lr 3.160854e-07 reg 2.362210e+04 train accuracy: 0.349122 val accuracy: 0.369000
lr 3.177632e-07 reg 5.065519e+04 train accuracy: 0.344776 val accuracy: 0.359000
lr 3.201699e-07 reg 5.226299e+04 train accuracy: 0.341408 val accuracy: 0.342000
lr 3.377372e-07 reg 2.269283e+03 train accuracy: 0.380449 val accuracy: 0.396000
lr 3.616771e-07 reg 6.938132e+04 train accuracy: 0.300286 val accuracy: 0.300000
lr 3.718385e-07 reg 1.983709e+04 train accuracy: 0.349857 val accuracy: 0.365000
lr 3.770505e-07 reg 3.780347e+04 train accuracy: 0.350939 val accuracy: 0.358000
lr 4.046878e-07 reg 3.371543e+03 train accuracy: 0.371388 val accuracy: 0.359000
lr 4.054919e-07 reg 1.239237e+04 train accuracy: 0.365490 val accuracy: 0.369000
lr 4.092683e-07 reg 2.666731e+03 train accuracy: 0.389837 val accuracy: 0.385000
lr 4.145726e-07 reg 3.265588e+04 train accuracy: 0.323490 val accuracy: 0.334000
lr 4.450604e-07 reg 1.679407e+03 train accuracy: 0.372367 val accuracy: 0.358000
lr 4.537898e-07 reg 2.225558e+03 train accuracy: 0.385102 val accuracy: 0.381000
lr 4.919736e-07 reg 1.224771e+04 train accuracy: 0.357143 val accuracy: 0.371000
lr 5.183424e-07 reg 6.859811e+03 train accuracy: 0.364694 val accuracy: 0.361000
lr 5.517516e-07 reg 2.243773e+04 train accuracy: 0.338633 val accuracy: 0.344000
lr 5.564994e-07 reg 7.732695e+03 train accuracy: 0.353143 val accuracy: 0.360000
lr 5.600230e-07 reg 1.082113e+04 train accuracy: 0.333102 val accuracy: 0.336000
lr 6.196379e-07 reg 1.094974e+03 train accuracy: 0.382388 val accuracy: 0.360000
lr 6.200040e-07 reg 3.314158e+03 train accuracy: 0.362184 val accuracy: 0.375000
lr 6.468560e-07 reg 3.004239e+03 train accuracy: 0.359224 val accuracy: 0.379000
lr 6.522689e-07 reg 1.343565e+04 train accuracy: 0.344163 val accuracy: 0.355000
lr 6.914719e-07 reg 8.582505e+03 train accuracy: 0.356510 val accuracy: 0.371000
lr 7.006970e-07 reg 5.514202e+03 train accuracy: 0.351286 val accuracy: 0.355000
lr 7.106459e-07 reg 1.954337e+04 train accuracy: 0.335184 val accuracy: 0.367000
lr 7.406458e-07 reg 1.710929e+03 train accuracy: 0.373918 val accuracy: 0.385000
lr 7.427225e-07 reg 2.189724e+03 train accuracy: 0.362367 val accuracy: 0.379000
lr 7.564386e-07 reg 4.088675e+03 train accuracy: 0.348939 val accuracy: 0.350000
lr 7.591422e-07 reg 7.633858e+04 train accuracy: 0.281469 val accuracy: 0.258000
lr 7.599919e-07 reg 3.400077e+04 train accuracy: 0.285633 val accuracy: 0.290000
lr 7.646597e-07 reg 2.218757e+04 train accuracy: 0.332102 val accuracy: 0.328000
lr 7.706970e-07 reg 5.680099e+04 train accuracy: 0.274755 val accuracy: 0.274000
lr 7.888968e-07 reg 8.559740e+03 train accuracy: 0.322857 val accuracy: 0.335000
lr 8.024898e-07 reg 8.819728e+03 train accuracy: 0.330694 val accuracy: 0.356000
lr 8.127446e-07 reg 2.775644e+04 train accuracy: 0.314020 val accuracy: 0.328000
lr 8.228903e-07 reg 4.726419e+03 train accuracy: 0.352184 val accuracy: 0.361000
lr 8.616139e-07 reg 8.678277e+03 train accuracy: 0.340612 val accuracy: 0.350000
lr 8.928070e-07 reg 5.761996e+04 train accuracy: 0.290980 val accuracy: 0.322000
lr 9.079893e-07 reg 2.347067e+04 train accuracy: 0.307551 val accuracy: 0.320000
lr 9.436775e-07 reg 4.632632e+03 train accuracy: 0.348510 val accuracy: 0.364000
lr 1.001401e-06 reg 6.525055e+03 train accuracy: 0.340714 val accuracy: 0.343000
lr 1.005986e-06 reg 4.368736e+04 train accuracy: 0.277184 val accuracy: 0.287000
lr 1.012448e-06 reg 2.473054e+03 train accuracy: 0.308612 val accuracy: 0.300000
lr 1.060057e-06 reg 4.778034e+03 train accuracy: 0.291224 val accuracy: 0.318000
lr 1.084912e-06 reg 9.234541e+03 train accuracy: 0.292510 val accuracy: 0.295000
lr 1.086625e-06 reg 8.025589e+03 train accuracy: 0.303714 val accuracy: 0.313000
lr 1.116463e-06 reg 1.969507e+03 train accuracy: 0.361367 val accuracy: 0.364000
lr 1.129951e-06 reg 1.927186e+04 train accuracy: 0.271102 val accuracy: 0.277000
lr 1.146565e-06 reg 3.465746e+04 train accuracy: 0.257612 val accuracy: 0.291000
lr 1.158538e-06 reg 1.375353e+03 train accuracy: 0.326184 val accuracy: 0.341000
lr 1.237830e-06 reg 5.888692e+03 train accuracy: 0.292000 val accuracy: 0.289000
lr 1.324854e-06 reg 1.874488e+04 train accuracy: 0.252449 val accuracy: 0.268000
lr 1.363163e-06 reg 4.135475e+03 train accuracy: 0.308816 val accuracy: 0.317000
lr 1.467563e-06 reg 2.265984e+04 train accuracy: 0.267796 val accuracy: 0.268000
lr 1.477311e-06 reg 3.539661e+04 train accuracy: 0.272612 val accuracy: 0.277000
lr 1.514845e-06 reg 1.100860e+04 train accuracy: 0.274265 val accuracy: 0.266000
lr 1.535611e-06 reg 1.338939e+03 train accuracy: 0.306429 val accuracy: 0.313000
lr 1.673671e-06 reg 1.701783e+03 train accuracy: 0.269633 val accuracy: 0.274000
lr 1.717657e-06 reg 2.170477e+04 train accuracy: 0.275510 val accuracy: 0.277000
lr 1.800612e-06 reg 3.847695e+03 train accuracy: 0.273816 val accuracy: 0.278000
lr 1.825563e-06 reg 9.799036e+03 train accuracy: 0.274327 val accuracy: 0.264000
lr 1.855326e-06 reg 1.614678e+04 train accuracy: 0.242286 val accuracy: 0.223000
lr 1.877207e-06 reg 9.737161e+04 train accuracy: 0.177571 val accuracy: 0.175000
lr 1.912774e-06 reg 6.132837e+04 train accuracy: 0.207878 val accuracy: 0.227000
lr 1.926301e-06 reg 8.352340e+04 train accuracy: 0.224102 val accuracy: 0.218000
lr 1.980226e-06 reg 4.064680e+04 train accuracy: 0.236551 val accuracy: 0.226000
lr 2.078896e-06 reg 2.789719e+03 train accuracy: 0.294000 val accuracy: 0.325000
lr 2.154519e-06 reg 4.442306e+03 train accuracy: 0.286612 val accuracy: 0.272000
lr 2.191779e-06 reg 1.071029e+04 train accuracy: 0.241429 val accuracy: 0.266000
lr 2.374718e-06 reg 7.182963e+03 train accuracy: 0.277388 val accuracy: 0.295000
lr 2.841418e-06 reg 2.794774e+04 train accuracy: 0.193592 val accuracy: 0.206000
lr 3.031753e-06 reg 5.209858e+03 train accuracy: 0.252102 val accuracy: 0.249000
lr 3.077343e-06 reg 1.527029e+04 train accuracy: 0.231939 val accuracy: 0.222000
lr 3.098334e-06 reg 1.021529e+04 train accuracy: 0.238592 val accuracy: 0.245000
lr 3.225751e-06 reg 7.308369e+03 train accuracy: 0.215898 val accuracy: 0.218000
lr 3.330940e-06 reg 4.406157e+03 train accuracy: 0.283694 val accuracy: 0.303000
lr 3.387889e-06 reg 3.460715e+03 train accuracy: 0.284735 val accuracy: 0.275000
lr 3.674137e-06 reg 5.184820e+03 train accuracy: 0.232918 val accuracy: 0.245000
lr 4.157191e-06 reg 3.025813e+04 train accuracy: 0.204980 val accuracy: 0.213000
best validation accuracy achieved during cross-validation: 0.396000
```

[71]:
```python
# Visualize the cross-validation results
import math
import pdb

# pdb.set_trace()

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.tight_layout(pad=3)
plt.scatter(x_scatter, y_scatter, marker_size, c=colors, cmap=plt.cm.coolwarm)
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 is 20
plt.subplot(2, 1, 2)
plt.scatter(x_scatter, y_scatter, marker_size, c=colors, cmap=plt.cm.coolwarm)
plt.colorbar()
plt.xlabel('log learning rate')
plt.ylabel('log regularization strength')
plt.title('CIFAR-10 validation accuracy')
plt.show()
```



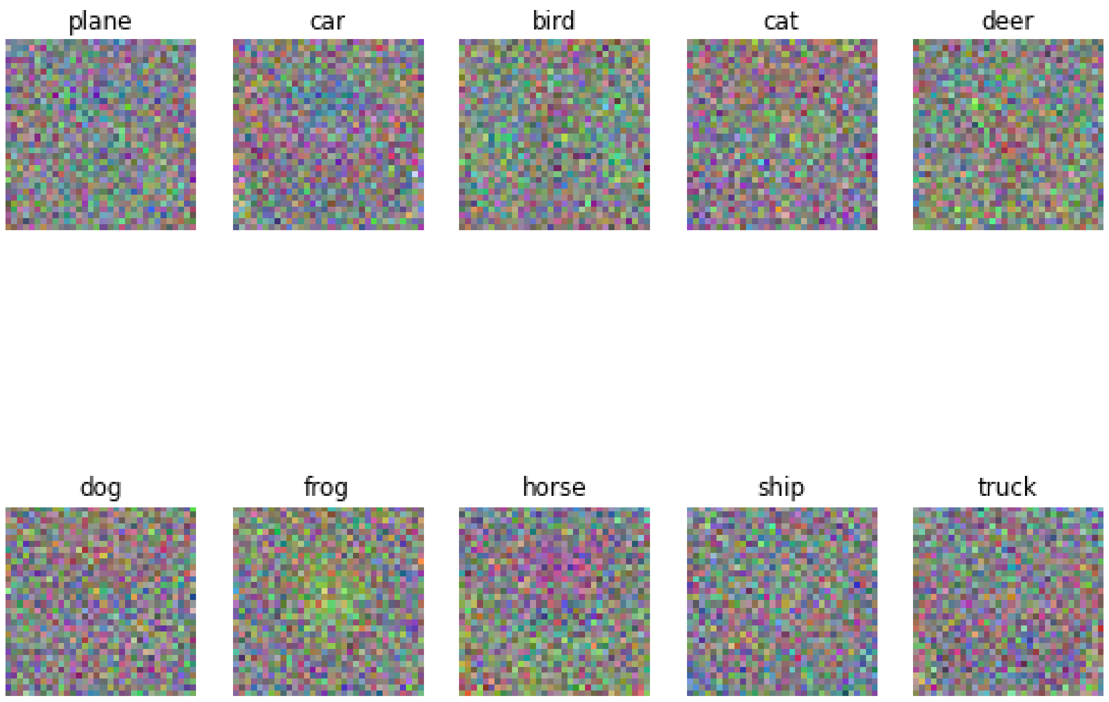


[72]:
```
# Evaluate the best svm on test set
y_test_pred = best_svm.predict(X_test)
test_accuracy = np.mean(y_test == y_test_pred)
print('linear SVM on raw pixels final test set accuracy: %f' % test_accuracy)
```

```
linear SVM on raw pixels final test set accuracy: 0.381000
```

[73]:
```
# Visualize the learned weights for each class.
# Depending on your choice of learning rate and regularization strength, 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', '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])
```



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 : We can observe some tendencies of the colors of the pixels. For example, we can see the cluster of green-colored pixels for frog. Therefore, if we have a picture that has a green matter in the center, that picture will get a high score for the frog class.

softmax

August 6, 2021

[ ]:
```
# This mounts your Google Drive to the Colab VM.
from google.colab import drive
drive.mount('/content/drive', force_remount=True)

# Enter the foldername in your Drive where you have saved the unzipped
# assignment folder, e.g. 'cs231n/assignments/assignment1/'
FOLDERNAME = 'CS231N/assignment1/'
assert FOLDERNAME is not None, "[!] Enter the foldername."

# Now that we've mounted your Drive, this ensures that
# the Python interpreter of the Colab VM can load
# python files from within it.
import sys
sys.path.append('/content/drive/My Drive/{}'.format(FOLDERNAME))

# This downloads the CIFAR-10 dataset to your Drive
# if it doesn't already exist.
%cd drive/My\ Drive/$FOLDERNAME/cs231n/datasets/
!bash get_datasets.sh
%cd /content/drive/My\ Drive/$FOLDERNAME
```

```
Mounted at /content/drive
/content/drive/My Drive/CS231N/assignment1/cs231n/datasets
/content/drive/My Drive/CS231N/assignment1
```

1 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 *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

[21]:
```
import random
import numpy as np
from cs231n.data_utils import load_CIFAR10
import matplotlib.pyplot as plt

%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-ipython
%load_ext autoreload
%autoreload 2
```

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

[22]:
```
def get_CIFAR10_data(num_training=49000, num_validation=1000, num_test=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 = 'cs231n/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)

    # 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_dev


# 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_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)
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,)
```

1.1 Softmax Classifier

Your code for this section will all be written inside `cs231n/classifiers/softmax.py`.

[23]:
```
# First implement the naive softmax loss function with nested loops.
# Open the file cs231n/classifiers/softmax.py and implement the
# softmax_loss_naive function.

from cs231n.classifiers.softmax import softmax_loss_naive
import time

# Generate a random softmax weight matrix and use it to compute the loss.
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: 2.412196
sanity check: 2.302585
```

Inline Question 1

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

Your Answer : If we didn't train the classifier and the classes of given test set are randomly distributed, we can expect that the probability to have a correct answer is 0.1 since there are 10 classes. Therefore, our loss will be close to -log(0.1).

[24]:
```
# 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 cs231n.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: 1.219294 analytic: 1.219294, relative error: 1.336925e-08
numerical: 1.777633 analytic: 1.777633, relative error: 1.841563e-08
numerical: -0.966367 analytic: -0.966367, relative error: 2.663353e-09
numerical: -1.302920 analytic: -1.302920, relative error: 2.347567e-08
numerical: -0.065044 analytic: -0.065044, relative error: 2.557012e-07
numerical: 2.093191 analytic: 2.093191, relative error: 1.164631e-08
numerical: 3.094625 analytic: 3.094625, relative error: 1.076320e-08
numerical: 1.561059 analytic: 1.561059, relative error: 9.405059e-09
numerical: 0.210860 analytic: 0.210860, relative error: 5.416631e-08
numerical: 0.394173 analytic: 0.394173, relative error: 1.366575e-08
numerical: 0.771407 analytic: 0.771407, relative error: 4.323798e-08
numerical: 2.386205 analytic: 2.386205, relative error: 4.079642e-09
numerical: -3.528928 analytic: -3.528928, relative error: 4.888060e-09
numerical: 1.414188 analytic: 1.414188, relative error: 8.016928e-09
numerical: 0.001498 analytic: 0.001498, relative error: 4.398412e-05
numerical: -0.658425 analytic: -0.658425, relative error: 2.289191e-08
numerical: 1.380203 analytic: 1.380203, relative error: 6.296310e-09
numerical: 1.057905 analytic: 1.057905, relative error: 3.689526e-09
numerical: 1.274182 analytic: 1.274182, relative error: 8.604773e-09
numerical: 2.826451 analytic: 2.826450, relative error: 2.448241e-08
```

[27]:
```
# Now that we have a naive implementation of the softmax loss function and its␣
↪gradient,
# implement a vectorized version in softmax_loss_vectorized.
# The two versions should compute the same results, but the vectorized 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 cs231n.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='fro')
print('Loss difference: %f' % np.abs(loss_naive - loss_vectorized))
print('Gradient difference: %f' % grad_difference)
```

```
naive loss: 2.412196e+00 computed in 0.141052s
vectorized loss: 2.412196e+00 computed in 0.014179s
```

```
Loss difference: 0.000000
Gradient difference: 0.000000
```

[32]:
```
# Use the validation set to tune hyperparameters (regularization strength and
# learning rate). You should experiment with different ranges for the learning
# rates and regularization strengths; if you are careful you should be able to
# get a classification accuracy of over 0.35 on the validation set.

from cs231n.classifiers import Softmax
results = {}
best_val = -1
best_softmax = 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 softmax classifer in best_softmax.                          ␣
 ↪#
################################################################################

# Provided as a reference. You may or may not want to change these␣
 ↪hyperparameters
learning_rates = [1e-7, 5e-7]
regularization_strengths = [2.5e4, 5e4]

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

max_count = 600
for count in range(max_count):
  print("count {} / {}".format(count, max_count))
  lr = 10**np.random.uniform(-9, -5)
  rs = 10**np.random.uniform(-1, 5)

  test_softmax = Softmax()
  test_loss = test_softmax.train(X_train, y_train, learning_rate=lr, reg=rs,
                  num_iters=20, verbose=True)
  if test_loss[-1] < 12.5:
    real_softmax = Softmax()
    real_softmax.train(X_train, y_train, learning_rate=lr, reg=rs,
                  num_iters=1500, verbose=True)

    y_val_pred = real_softmax.predict(X_val)
    val_accuracy = np.mean(y_val == y_val_pred)
    y_train_pred = real_softmax.predict(X_train)
    train_accuracy = np.mean(y_train == y_train_pred)
    results[(lr, rs)] = (train_accuracy, val_accuracy)

    if val_accuracy > best_val:
      best_val = val_accuracy
      best_softmax = real_softmax
      print("best_val updated to %f" %val_accuracy)

    else:
      print("Val_accuracy: {}, Current best_val: {}".format(val_accuracy,␣
 ↪best_val))

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

# 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)
```

```
     5000 .
iteration 300 / 1500: loss 2.857647
iteration 400 / 1500: loss 3.346306
iteration 500 / 1500: loss 2.777049
iteration 600 / 1500: loss 3.062846
iteration 700 / 1500: loss 2.373326
iteration 800 / 1500: loss 2.769278
iteration 900 / 1500: loss 2.716033
iteration 1000 / 1500: loss 2.261531
iteration 1100 / 1500: loss 2.279089
iteration 1200 / 1500: loss 2.386857
iteration 1300 / 1500: loss 2.136848
iteration 1400 / 1500: loss 2.265926
Val_accuracy: 0.345, Current best_val: 0.4
count 179 / 600
iteration 0 / 20: loss 6.929913
iteration 0 / 1500: loss 7.777618
iteration 100 / 1500: loss 4.216967
iteration 200 / 1500: loss 4.189327
iteration 300 / 1500: loss 3.857015
iteration 400 / 1500: loss 3.737713
iteration 500 / 1500: loss 3.444041
iteration 600 / 1500: loss 3.434889
iteration 700 / 1500: loss 3.265382
iteration 800 / 1500: loss 3.442471
iteration 900 / 1500: loss 3.271451
iteration 1000 / 1500: loss 3.228809
iteration 1100 / 1500: loss 3.332584
iteration 1200 / 1500: loss 3.115540
iteration 1300 / 1500: loss 3.080722
iteration 1400 / 1500: loss 3.150541
Val_accuracy: 0.351, Current best_val: 0.4
count 180 / 600
iteration 0 / 20: loss 166.883951
count 181 / 600
iteration 0 / 20: loss 580.146517
count 182 / 600
iteration 0 / 20: loss 2682.355553
count 183 / 600
iteration 0 / 20: loss 18.105518
count 184 / 600
iteration 0 / 20: loss 245.524717
count 185 / 600
iteration 0 / 20: loss 39.321068
count 186 / 600
iteration 0 / 20: loss 206.160370
count 187 / 600
iteration 0 / 20: loss 5.768267
iteration 0 / 1500: loss 5.238654
iteration 100 / 1500: loss 4.519078
iteration 200 / 1500: loss 4.118151
iteration 300 / 1500: loss 3.708981
iteration 400 / 1500: loss 3.648704
iteration 500 / 1500: loss 3.198759
iteration 600 / 1500: loss 3.410642
iteration 700 / 1500: loss 3.399851
iteration 800 / 1500: loss 3.056529
iteration 900 / 1500: loss 3.284960
iteration 1000 / 1500: loss 3.108982
iteration 1100 / 1500: loss 3.185188
iteration 1200 / 1500: loss 2.913982
iteration 1300 / 1500: loss 3.206562
iteration 1400 / 1500: loss 3.026181
Val_accuracy: 0.271, Current best_val: 0.4
count 188 / 600
iteration 0 / 20: loss 7.160637
iteration 0 / 1500: loss 6.722282
iteration 100 / 1500: loss 5.302564
iteration 200 / 1500: loss 4.883048
iteration 300 / 1500: loss 4.607078
iteration 400 / 1500: loss 4.365610
iteration 500 / 1500: loss 4.198793
iteration 600 / 1500: loss 4.235878
iteration 700 / 1500: loss 4.108563
iteration 800 / 1500: loss 3.992678
iteration 900 / 1500: loss 4.340014
iteration 1000 / 1500: loss 3.978418
iteration 1100 / 1500: loss 3.846935
iteration 1200 / 1500: loss 3.974962
iteration 1300 / 1500: loss 3.734787
iteration 1400 / 1500: loss 3.665059
Val_accuracy: 0.27, Current best_val: 0.4
count 189 / 600
iteration 0 / 20: loss 6.045495
iteration 0 / 1500: loss 6.051478
iteration 100 / 1500: loss 5.752438
iteration 200 / 1500: loss 5.683434
iteration 300 / 1500: loss 6.490300
iteration 400 / 1500: loss 5.628458
iteration 500 / 1500: loss 5.815076
iteration 600 / 1500: loss 6.056241
iteration 700 / 1500: loss 5.559779
iteration 800 / 1500: loss 5.429185
iteration 900 / 1500: loss 5.819560
iteration 1000 / 1500: loss 5.335129
iteration 1100 / 1500: loss 5.407219
iteration 1200 / 1500: loss 5.520642
iteration 1300 / 1500: loss 5.684749
iteration 1400 / 1500: loss 5.499404
Val_accuracy: 0.088, Current best_val: 0.4
count 190 / 600
iteration 0 / 20: loss 5.656817
iteration 0 / 1500: loss 5.732047
iteration 100 / 1500: loss 5.958650
iteration 200 / 1500: loss 5.587838
iteration 300 / 1500: loss 5.914406
iteration 400 / 1500: loss 5.617174
iteration 500 / 1500: loss 4.955954
iteration 600 / 1500: loss 5.522031
iteration 700 / 1500: loss 5.247080
iteration 800 / 1500: loss 5.110322
iteration 900 / 1500: loss 4.795549
iteration 1000 / 1500: loss 4.647294
iteration 1100 / 1500: loss 5.169993
iteration 1200 / 1500: loss 5.406743
iteration 1300 / 1500: loss 4.735059
iteration 1400 / 1500: loss 4.859245
Val_accuracy: 0.143, Current best_val: 0.4
count 191 / 600
```

```
iteration 0 / 20: loss 1102.724059
count 192 / 600
iteration 0 / 20: loss 726.425160
iteration 0 / 1500: loss 727.902866
iteration 100 / 1500: loss 2.264766
iteration 200 / 1500: loss 2.229221
iteration 300 / 1500: loss 2.082668
iteration 400 / 1500: loss 2.112064
iteration 500 / 1500: loss 2.166778
iteration 600 / 1500: loss 2.217382
iteration 700 / 1500: loss 2.121636
iteration 800 / 1500: loss 2.174234
iteration 900 / 1500: loss 2.242197
iteration 1000 / 1500: loss 2.282542
iteration 1100 / 1500: loss 2.126417
iteration 1200 / 1500: loss 2.198466
iteration 1300 / 1500: loss 2.162335
iteration 1400 / 1500: loss 2.147797
Val_accuracy: 0.265, Current best_val: 0.4
count 193 / 600
iteration 0 / 20: loss 56.092142
count 194 / 600
iteration 0 / 20: loss 252.293965
count 195 / 600
iteration 0 / 20: loss 13.462683
iteration 0 / 1500: loss 13.493981
iteration 100 / 1500: loss 5.618447
iteration 200 / 1500: loss 4.180632
iteration 300 / 1500: loss 4.214483
iteration 400 / 1500: loss 2.965096
iteration 500 / 1500: loss 3.193404
iteration 600 / 1500: loss 4.141981
iteration 700 / 1500: loss 3.886358
iteration 800 / 1500: loss 2.783051
iteration 900 / 1500: loss 2.111513
iteration 1000 / 1500: loss 2.571427
iteration 1100 / 1500: loss 2.808615
iteration 1200 / 1500: loss 3.411574
iteration 1300 / 1500: loss 2.835155
iteration 1400 / 1500: loss 3.096898
Val_accuracy: 0.258, Current best_val: 0.4
count 196 / 600
iteration 0 / 20: loss 10.027065
iteration 0 / 1500: loss 10.607394
iteration 100 / 1500: loss 8.408860
iteration 200 / 1500: loss 8.016483
iteration 300 / 1500: loss 7.646445
iteration 400 / 1500: loss 7.488562
iteration 500 / 1500: loss 7.316091
iteration 600 / 1500: loss 7.125584
iteration 700 / 1500: loss 7.336990
iteration 800 / 1500: loss 7.234587
iteration 900 / 1500: loss 6.957870
iteration 1000 / 1500: loss 7.038301
iteration 1100 / 1500: loss 6.913277
iteration 1200 / 1500: loss 6.841683
iteration 1300 / 1500: loss 6.617075
iteration 1400 / 1500: loss 6.544865
Val_accuracy: 0.264, Current best_val: 0.4
count 197 / 600
iteration 0 / 20: loss 12.601525
iteration 0 / 1500: loss 12.463376
iteration 100 / 1500: loss 10.849283
iteration 200 / 1500: loss 10.481661
iteration 300 / 1500: loss 9.839386
iteration 400 / 1500: loss 9.512935
iteration 500 / 1500: loss 9.307905
iteration 600 / 1500: loss 9.258879
iteration 700 / 1500: loss 9.240475
iteration 800 / 1500: loss 8.980344
iteration 900 / 1500: loss 9.209645
iteration 1000 / 1500: loss 8.914706
iteration 1100 / 1500: loss 8.757186
iteration 1200 / 1500: loss 8.800790
iteration 1300 / 1500: loss 8.833891
iteration 1400 / 1500: loss 8.658883
Val_accuracy: 0.256, Current best_val: 0.4
count 198 / 600
iteration 0 / 20: loss 7.200432
iteration 0 / 1500: loss 7.030178
iteration 100 / 1500: loss 4.775421
iteration 200 / 1500: loss 4.357535
iteration 300 / 1500: loss 4.401869
iteration 400 / 1500: loss 3.897370
iteration 500 / 1500: loss 3.914940
iteration 600 / 1500: loss 3.983700
iteration 700 / 1500: loss 3.779706
iteration 800 / 1500: loss 3.717354
iteration 900 / 1500: loss 3.619998
iteration 1000 / 1500: loss 3.497026
iteration 1100 / 1500: loss 3.635179
iteration 1200 / 1500: loss 3.652375
iteration 1300 / 1500: loss 3.442223
iteration 1400 / 1500: loss 3.551922
Val_accuracy: 0.321, Current best_val: 0.4
count 199 / 600
```

```
iteration 0 / 20: loss 174.602278
count 200 / 600
iteration 0 / 20: loss 1547.587621
count 201 / 600
iteration 0 / 20: loss 10.830293
iteration 0 / 1500: loss 9.828819
iteration 100 / 1500: loss 9.997442
iteration 200 / 1500: loss 10.256320
iteration 300 / 1500: loss 9.744840
iteration 400 / 1500: loss 9.943300
iteration 500 / 1500: loss 10.015639
iteration 600 / 1500: loss 9.881103
iteration 700 / 1500: loss 9.723183
iteration 800 / 1500: loss 9.636373
iteration 900 / 1500: loss 9.700375
iteration 1000 / 1500: loss 9.402956
iteration 1100 / 1500: loss 9.537045
iteration 1200 / 1500: loss 9.466210
iteration 1300 / 1500: loss 9.896635
iteration 1400 / 1500: loss 9.451583
Val_accuracy: 0.119, Current best_val: 0.4
count 202 / 600
iteration 0 / 20: loss 5.871141
iteration 0 / 1500: loss 5.727747
iteration 100 / 1500: loss 2.285836
iteration 200 / 1500: loss 2.189020
iteration 300 / 1500: loss 2.072472
iteration 400 / 1500: loss 2.088666
iteration 500 / 1500: loss 2.145768
iteration 600 / 1500: loss 2.143583
iteration 700 / 1500: loss 2.157632
iteration 800 / 1500: loss 2.355592
iteration 900 / 1500: loss 2.295458
iteration 1000 / 1500: loss 2.357024
iteration 1100 / 1500: loss 2.057093
iteration 1200 / 1500: loss 2.219000
iteration 1300 / 1500: loss 2.085543
iteration 1400 / 1500: loss 1.945440
Val_accuracy: 0.363, Current best_val: 0.4
count 203 / 600
iteration 0 / 20: loss 6.199878
iteration 0 / 1500: loss 6.160018
iteration 100 / 1500: loss 6.375397
iteration 200 / 1500: loss 6.499444
iteration 300 / 1500: loss 6.405350
iteration 400 / 1500: loss 6.148181
iteration 500 / 1500: loss 6.022168
iteration 600 / 1500: loss 6.319868
iteration 700 / 1500: loss 6.063351
iteration 800 / 1500: loss 5.978072
iteration 900 / 1500: loss 5.771312
iteration 1000 / 1500: loss 6.136640
iteration 1100 / 1500: loss 6.210274
iteration 1200 / 1500: loss 6.346970
iteration 1300 / 1500: loss 6.036232
iteration 1400 / 1500: loss 6.137247
Val_accuracy: 0.119, Current best_val: 0.4
count 204 / 600
iteration 0 / 20: loss 29.459554
count 205 / 600
iteration 0 / 20: loss 12.928440
iteration 0 / 1500: loss 13.138702
iteration 100 / 1500: loss 5.712188
iteration 200 / 1500: loss 5.359477
iteration 300 / 1500: loss 3.175870
iteration 400 / 1500: loss 2.537648
iteration 500 / 1500: loss 2.301114
iteration 600 / 1500: loss 2.171245
iteration 700 / 1500: loss 2.543568
iteration 800 / 1500: loss 2.357645
iteration 900 / 1500: loss 2.310233
iteration 1000 / 1500: loss 2.201621
iteration 1100 / 1500: loss 2.677568
iteration 1200 / 1500: loss 2.183782
iteration 1300 / 1500: loss 2.396513
iteration 1400 / 1500: loss 2.506441
Val_accuracy: 0.323, Current best_val: 0.4
count 206 / 600
iteration 0 / 20: loss 193.428347
count 207 / 600
iteration 0 / 20: loss 5.941178
iteration 0 / 1500: loss 6.081333
iteration 100 / 1500: loss 3.166592
iteration 200 / 1500: loss 2.735603
iteration 300 / 1500: loss 2.142921
iteration 400 / 1500: loss 2.167269
iteration 500 / 1500: loss 2.074856
iteration 600 / 1500: loss 3.022501
iteration 700 / 1500: loss 2.236763
iteration 800 / 1500: loss 2.341341
iteration 900 / 1500: loss 2.358377
iteration 1000 / 1500: loss 2.678604
iteration 1100 / 1500: loss 2.351336
iteration 1200 / 1500: loss 2.488447
iteration 1300 / 1500: loss 2.118870
iteration 1400 / 1500: loss 2.146010
```

```
Val_accuracy: 0.302, Current best_val: 0.4
count 208 / 600
iteration 0 / 20: loss 5.644891
iteration 0 / 1500: loss 5.803381
iteration 100 / 1500: loss 3.070553
iteration 200 / 1500: loss 2.365498
iteration 300 / 1500: loss 2.231275
iteration 400 / 1500: loss 2.261198
iteration 500 / 1500: loss 2.180485
iteration 600 / 1500: loss 2.064034
iteration 700 / 1500: loss 2.057936
iteration 800 / 1500: loss 2.153326
iteration 900 / 1500: loss 1.927181
iteration 1000 / 1500: loss 2.000165
iteration 1100 / 1500: loss 1.969880
iteration 1200 / 1500: loss 1.980554
iteration 1300 / 1500: loss 2.017385
iteration 1400 / 1500: loss 2.067038
Val_accuracy: 0.327, Current best_val: 0.4
count 209 / 600
iteration 0 / 20: loss 6.750479
iteration 0 / 1500: loss 5.782702
iteration 100 / 1500: loss 5.721400
iteration 200 / 1500: loss 5.297010
iteration 300 / 1500: loss 5.907214
iteration 400 / 1500: loss 5.709820
iteration 500 / 1500: loss 5.916807
iteration 600 / 1500: loss 5.670923
iteration 700 / 1500: loss 4.877486
iteration 800 / 1500: loss 4.807923
iteration 900 / 1500: loss 5.791174
iteration 1000 / 1500: loss 5.265606
iteration 1100 / 1500: loss 5.795817
iteration 1200 / 1500: loss 5.856657
iteration 1300 / 1500: loss 5.258849
iteration 1400 / 1500: loss 4.605519
Val_accuracy: 0.113, Current best_val: 0.4
count 210 / 600
iteration 0 / 20: loss 64.518549
count 211 / 600
iteration 0 / 20: loss 123.244815
count 212 / 600
iteration 0 / 20: loss 5.882065
iteration 0 / 1500: loss 5.521655
iteration 100 / 1500: loss 2.357410
iteration 200 / 1500: loss 2.535310
iteration 300 / 1500: loss 2.342707
iteration 400 / 1500: loss 2.801877
iteration 500 / 1500: loss 2.039323
iteration 600 / 1500: loss 2.428116
iteration 700 / 1500: loss 2.176685
iteration 800 / 1500: loss 2.510962
iteration 900 / 1500: loss 2.118342
iteration 1000 / 1500: loss 2.047015
iteration 1100 / 1500: loss 1.955125
iteration 1200 / 1500: loss 2.591649
iteration 1300 / 1500: loss 1.912097
iteration 1400 / 1500: loss 1.835854
Val_accuracy: 0.324, Current best_val: 0.4
count 213 / 600
iteration 0 / 20: loss 6.011135
iteration 0 / 1500: loss 5.774653
iteration 100 / 1500: loss 5.939825
iteration 200 / 1500: loss 5.415841
iteration 300 / 1500: loss 5.244802
iteration 400 / 1500: loss 5.337655
iteration 500 / 1500: loss 5.094698
iteration 600 / 1500: loss 5.165694
iteration 700 / 1500: loss 4.968331
iteration 800 / 1500: loss 4.911541
iteration 900 / 1500: loss 4.852794
iteration 1000 / 1500: loss 4.962671
iteration 1100 / 1500: loss 4.708259
iteration 1200 / 1500: loss 4.318636
iteration 1300 / 1500: loss 4.906299
iteration 1400 / 1500: loss 4.578025
Val_accuracy: 0.111, Current best_val: 0.4
count 214 / 600
iteration 0 / 20: loss 5.696500
iteration 0 / 1500: loss 5.671132
iteration 100 / 1500: loss 5.627630
iteration 200 / 1500: loss 5.147057
iteration 300 / 1500: loss 4.744999
iteration 400 / 1500: loss 4.860497
iteration 500 / 1500: loss 4.100622
iteration 600 / 1500: loss 4.305850
iteration 700 / 1500: loss 4.107321
iteration 800 / 1500: loss 4.220094
iteration 900 / 1500: loss 3.859908
iteration 1000 / 1500: loss 4.166381
iteration 1100 / 1500: loss 3.844113
iteration 1200 / 1500: loss 3.895283
iteration 1300 / 1500: loss 3.738997
iteration 1400 / 1500: loss 3.316918
Val_accuracy: 0.175, Current best_val: 0.4
count 215 / 600
```

```
iteration 0 / 20: loss 5.553081
iteration 0 / 1500: loss 5.389252
iteration 100 / 1500: loss 5.526917
iteration 200 / 1500: loss 5.382885
iteration 300 / 1500: loss 5.199549
iteration 400 / 1500: loss 4.564232
iteration 500 / 1500: loss 5.263424
iteration 600 / 1500: loss 4.990462
iteration 700 / 1500: loss 4.792193
iteration 800 / 1500: loss 5.134116
iteration 900 / 1500: loss 5.005744
iteration 1000 / 1500: loss 4.919246
iteration 1100 / 1500: loss 4.985564
iteration 1200 / 1500: loss 4.561482
iteration 1300 / 1500: loss 4.729344
iteration 1400 / 1500: loss 4.587512
Val_accuracy: 0.135, Current best_val: 0.4
count 216 / 600
iteration 0 / 20: loss 2791.658605
count 217 / 600
iteration 0 / 20: loss 35.253826
count 218 / 600
iteration 0 / 20: loss 1256.464739
count 219 / 600
iteration 0 / 20: loss 5.834781
iteration 0 / 1500: loss 6.342425
iteration 100 / 1500: loss 3.494612
iteration 200 / 1500: loss 3.438203
iteration 300 / 1500: loss 3.319892
iteration 400 / 1500: loss 2.899275
iteration 500 / 1500: loss 2.876675
iteration 600 / 1500: loss 2.951674
iteration 700 / 1500: loss 2.842062
iteration 800 / 1500: loss 3.024118
iteration 900 / 1500: loss 2.692131
iteration 1000 / 1500: loss 2.470586
iteration 1100 / 1500: loss 2.610234
iteration 1200 / 1500: loss 2.650544
iteration 1300 / 1500: loss 2.737183
iteration 1400 / 1500: loss 2.559950
Val_accuracy: 0.286, Current best_val: 0.4
count 220 / 600
iteration 0 / 20: loss 7.802468
iteration 0 / 1500: loss 6.130571
iteration 100 / 1500: loss 5.021083
iteration 200 / 1500: loss 4.630237
iteration 300 / 1500: loss 4.168894
iteration 400 / 1500: loss 3.926743
iteration 500 / 1500: loss 4.379038
iteration 600 / 1500: loss 4.046481
iteration 700 / 1500: loss 4.210451
iteration 800 / 1500: loss 3.938831
iteration 900 / 1500: loss 3.938009
iteration 1000 / 1500: loss 4.051140
iteration 1100 / 1500: loss 4.025002
iteration 1200 / 1500: loss 3.937433
iteration 1300 / 1500: loss 3.743715
iteration 1400 / 1500: loss 3.648614
Val_accuracy: 0.278, Current best_val: 0.4
count 221 / 600
iteration 0 / 20: loss 5.503303
iteration 0 / 1500: loss 4.901523
iteration 100 / 1500: loss 5.015581
iteration 200 / 1500: loss 4.522536
iteration 300 / 1500: loss 4.398982
iteration 400 / 1500: loss 4.457404
iteration 500 / 1500: loss 4.565387
iteration 600 / 1500: loss 4.368645
iteration 700 / 1500: loss 4.378377
iteration 800 / 1500: loss 4.005516
iteration 900 / 1500: loss 3.991486
iteration 1000 / 1500: loss 4.107289
iteration 1100 / 1500: loss 4.270220
iteration 1200 / 1500: loss 3.871677
iteration 1300 / 1500: loss 4.277429
iteration 1400 / 1500: loss 4.408065
Val_accuracy: 0.157, Current best_val: 0.4
count 222 / 600
iteration 0 / 20: loss 5.958419
iteration 0 / 1500: loss 5.145961
iteration 100 / 1500: loss 4.559046
iteration 200 / 1500: loss 4.085911
iteration 300 / 1500: loss 3.575592
iteration 400 / 1500: loss 3.543233
iteration 500 / 1500: loss 3.473848
iteration 600 / 1500: loss 3.525346
iteration 700 / 1500: loss 3.290096
iteration 800 / 1500: loss 3.247970
iteration 900 / 1500: loss 2.727901
iteration 1000 / 1500: loss 3.292134
iteration 1100 / 1500: loss 2.803575
iteration 1200 / 1500: loss 2.766045
iteration 1300 / 1500: loss 2.835734
iteration 1400 / 1500: loss 2.626159
Val_accuracy: 0.218, Current best_val: 0.4
count 223 / 600
```

```
iteration 0 / 20: loss 5.954509
iteration 0 / 1500: loss 4.952083
iteration 100 / 1500: loss 3.410359
iteration 200 / 1500: loss 2.981236
iteration 300 / 1500: loss 2.602763
iteration 400 / 1500: loss 2.545710
iteration 500 / 1500: loss 2.598823
iteration 600 / 1500: loss 2.258633
iteration 700 / 1500: loss 2.451729
iteration 800 / 1500: loss 2.226887
iteration 900 / 1500: loss 2.458196
iteration 1000 / 1500: loss 2.401251
iteration 1100 / 1500: loss 2.144849
iteration 1200 / 1500: loss 2.138065
iteration 1300 / 1500: loss 2.143820
iteration 1400 / 1500: loss 2.175804
Val_accuracy: 0.297, Current best_val: 0.4
count 224 / 600
iteration 0 / 20: loss 13.650207
count 225 / 600
iteration 0 / 20: loss 1797.072811
count 226 / 600
iteration 0 / 20: loss 396.387451
count 227 / 600
iteration 0 / 20: loss 5.275324
iteration 0 / 1500: loss 6.981258
iteration 100 / 1500: loss 2.404999
iteration 200 / 1500: loss 2.152657
iteration 300 / 1500: loss 2.072778
iteration 400 / 1500: loss 1.974716
iteration 500 / 1500: loss 1.935516
iteration 600 / 1500: loss 1.990555
iteration 700 / 1500: loss 2.076249
iteration 800 / 1500: loss 1.871855
iteration 900 / 1500: loss 1.851484
iteration 1000 / 1500: loss 1.817732
iteration 1100 / 1500: loss 1.822638
iteration 1200 / 1500: loss 1.929377
iteration 1300 / 1500: loss 1.664014
iteration 1400 / 1500: loss 1.778406
Val_accuracy: 0.374, Current best_val: 0.4
count 228 / 600
iteration 0 / 20: loss 13.917540
iteration 0 / 1500: loss 14.511551
iteration 100 / 1500: loss 5.749970
iteration 200 / 1500: loss 3.249552
iteration 300 / 1500: loss 3.257304
iteration 400 / 1500: loss 2.878755
iteration 500 / 1500: loss 2.210587
iteration 600 / 1500: loss 2.229333
iteration 700 / 1500: loss 2.538844
iteration 800 / 1500: loss 2.616649
iteration 900 / 1500: loss 3.480650
iteration 1000 / 1500: loss 2.719402
iteration 1100 / 1500: loss 2.362211
iteration 1200 / 1500: loss 3.328959
iteration 1300 / 1500: loss 2.073972
iteration 1400 / 1500: loss 2.782946
Val_accuracy: 0.327, Current best_val: 0.4
count 229 / 600
iteration 0 / 20: loss 4.724112
iteration 0 / 1500: loss 5.802916
iteration 100 / 1500: loss 3.705575
iteration 200 / 1500: loss 3.112080
iteration 300 / 1500: loss 3.137734
iteration 400 / 1500: loss 2.731329
iteration 500 / 1500: loss 2.796651
iteration 600 / 1500: loss 2.736017
iteration 700 / 1500: loss 2.684620
iteration 800 / 1500: loss 2.718253
iteration 900 / 1500: loss 2.691179
iteration 1000 / 1500: loss 2.483294
iteration 1100 / 1500: loss 2.534897
iteration 1200 / 1500: loss 2.711663
iteration 1300 / 1500: loss 2.564855
iteration 1400 / 1500: loss 2.400238
Val_accuracy: 0.294, Current best_val: 0.4
count 230 / 600
iteration 0 / 20: loss 91.420532
count 231 / 600
iteration 0 / 20: loss 4.823298
iteration 0 / 1500: loss 5.921021
iteration 100 / 1500: loss 6.017361
iteration 200 / 1500: loss 5.607036
iteration 300 / 1500: loss 5.588254
iteration 400 / 1500: loss 5.115680
iteration 500 / 1500: loss 4.811239
iteration 600 / 1500: loss 4.517202
iteration 700 / 1500: loss 4.612469
iteration 800 / 1500: loss 4.606047
iteration 900 / 1500: loss 4.472256
iteration 1000 / 1500: loss 4.301773
iteration 1100 / 1500: loss 4.602495
iteration 1200 / 1500: loss 4.420362
iteration 1300 / 1500: loss 3.948679
iteration 1400 / 1500: loss 4.060881
```

```
Val_accuracy: 0.15, Current best_val: 0.4
count 232 / 600
iteration 0 / 20: loss 8.024155
iteration 0 / 1500: loss 8.112336
iteration 100 / 1500: loss 5.724005
iteration 200 / 1500: loss 4.840398
iteration 300 / 1500: loss 4.937233
iteration 400 / 1500: loss 4.889383
iteration 500 / 1500: loss 4.547301
iteration 600 / 1500: loss 4.602873
iteration 700 / 1500: loss 4.489439
iteration 800 / 1500: loss 4.246041
iteration 900 / 1500: loss 4.334467
iteration 1000 / 1500: loss 4.265137
iteration 1100 / 1500: loss 4.100153
iteration 1200 / 1500: loss 4.127472
iteration 1300 / 1500: loss 3.852307
iteration 1400 / 1500: loss 3.810511
Val_accuracy: 0.332, Current best_val: 0.4
count 233 / 600
iteration 0 / 20: loss 5.840736
iteration 0 / 1500: loss 5.302894
iteration 100 / 1500: loss 5.232541
iteration 200 / 1500: loss 5.410345
iteration 300 / 1500: loss 4.710056
iteration 400 / 1500: loss 4.911604
iteration 500 / 1500: loss 4.543262
iteration 600 / 1500: loss 4.563080
iteration 700 / 1500: loss 4.529035
iteration 800 / 1500: loss 4.319458
iteration 900 / 1500: loss 4.036982
iteration 1000 / 1500: loss 4.134690
iteration 1100 / 1500: loss 4.239568
iteration 1200 / 1500: loss 4.384680
iteration 1300 / 1500: loss 4.037186
iteration 1400 / 1500: loss 3.987720
Val_accuracy: 0.129, Current best_val: 0.4
count 234 / 600
iteration 0 / 20: loss 14.061398
count 235 / 600
iteration 0 / 20: loss 6.851538
iteration 0 / 1500: loss 7.197013
iteration 100 / 1500: loss 7.057215
iteration 200 / 1500: loss 7.099220
iteration 300 / 1500: loss 6.771310
iteration 400 / 1500: loss 6.613397
iteration 500 / 1500: loss 6.303449
iteration 600 / 1500: loss 6.219102
iteration 700 / 1500: loss 6.308173
iteration 800 / 1500: loss 5.962868
iteration 900 / 1500: loss 5.751152
iteration 1000 / 1500: loss 5.796604
iteration 1100 / 1500: loss 5.659842
iteration 1200 / 1500: loss 5.379603
iteration 1300 / 1500: loss 5.044672
iteration 1400 / 1500: loss 5.299214
Val_accuracy: 0.152, Current best_val: 0.4
count 236 / 600
iteration 0 / 20: loss 295.730619
count 237 / 600
iteration 0 / 20: loss 86.897515
count 238 / 600
iteration 0 / 20: loss 5.380662
iteration 0 / 1500: loss 5.231796
iteration 100 / 1500: loss 3.700501
iteration 200 / 1500: loss 3.182407
iteration 300 / 1500: loss 3.089911
iteration 400 / 1500: loss 2.793625
iteration 500 / 1500: loss 2.849139
iteration 600 / 1500: loss 2.606322
iteration 700 / 1500: loss 2.701252
iteration 800 / 1500: loss 2.587037
iteration 900 / 1500: loss 2.651769
iteration 1000 / 1500: loss 2.559503
iteration 1100 / 1500: loss 2.771164
iteration 1200 / 1500: loss 2.281337
iteration 1300 / 1500: loss 2.429484
iteration 1400 / 1500: loss 2.290348
Val_accuracy: 0.297, Current best_val: 0.4
count 239 / 600
iteration 0 / 20: loss 18.161979
count 240 / 600
iteration 0 / 20: loss 15.146240
count 241 / 600
iteration 0 / 20: loss 50.547989
count 242 / 600
iteration 0 / 20: loss 88.364046
count 243 / 600
iteration 0 / 20: loss 19.167903
count 244 / 600
iteration 0 / 20: loss 5.876436
iteration 0 / 1500: loss 6.375515
iteration 100 / 1500: loss 5.280482
iteration 200 / 1500: loss 4.854195
iteration 300 / 1500: loss 4.618502
iteration 400 / 1500: loss 4.459331
iteration 500 / 1500: loss 4.536196
iteration 600 / 1500: loss 4.293712
iteration 700 / 1500: loss 4.099510
iteration 800 / 1500: loss 4.193131
iteration 900 / 1500: loss 3.987955
iteration 1000 / 1500: loss 3.860466
iteration 1100 / 1500: loss 3.851793
iteration 1200 / 1500: loss 4.031224
iteration 1300 / 1500: loss 3.831329
iteration 1400 / 1500: loss 3.823870
Val_accuracy: 0.178, Current best_val: 0.4
count 245 / 600
iteration 0 / 20: loss 6.177798
iteration 0 / 1500: loss 6.616489
iteration 100 / 1500: loss 3.621611
iteration 200 / 1500: loss 3.386546
iteration 300 / 1500: loss 3.276096
iteration 400 / 1500: loss 3.204182
iteration 500 / 1500: loss 3.013206
iteration 600 / 1500: loss 2.900062
iteration 700 / 1500: loss 2.976610
iteration 800 / 1500: loss 2.744894
iteration 900 / 1500: loss 2.912976
iteration 1000 / 1500: loss 2.675068
iteration 1100 / 1500: loss 2.680252
iteration 1200 / 1500: loss 2.443772
iteration 1300 / 1500: loss 2.414720
iteration 1400 / 1500: loss 2.586424
Val_accuracy: 0.372, Current best_val: 0.4
count 246 / 600
iteration 0 / 20: loss 750.770754
count 247 / 600
iteration 0 / 20: loss 6.782673
iteration 0 / 1500: loss 5.407639
iteration 100 / 1500: loss 4.703971
iteration 200 / 1500: loss 4.452993
iteration 300 / 1500: loss 4.334040
iteration 400 / 1500: loss 4.344763
iteration 500 / 1500: loss 3.910610
iteration 600 / 1500: loss 3.972924
iteration 700 / 1500: loss 3.825151
iteration 800 / 1500: loss 3.807316
iteration 900 / 1500: loss 4.214200
iteration 1000 / 1500: loss 3.610432
iteration 1100 / 1500: loss 3.349509
iteration 1200 / 1500: loss 3.311992
iteration 1300 / 1500: loss 3.244032
iteration 1400 / 1500: loss 3.463101
```

```
Val_accuracy: 0.175, Current best_val: 0.4
count 248 / 600
iteration 0 / 20: loss 5.841408
iteration 0 / 1500: loss 6.576766
iteration 100 / 1500: loss 5.692824
iteration 200 / 1500: loss 4.717242
iteration 300 / 1500: loss 4.746641
iteration 400 / 1500: loss 4.713414
iteration 500 / 1500: loss 4.522685
iteration 600 / 1500: loss 4.444355
iteration 700 / 1500: loss 4.381655
iteration 800 / 1500: loss 4.021559
iteration 900 / 1500: loss 4.061996
iteration 1000 / 1500: loss 4.305267
iteration 1100 / 1500: loss 3.840293
iteration 1200 / 1500: loss 3.973998
iteration 1300 / 1500: loss 3.884037
iteration 1400 / 1500: loss 3.967048
Val_accuracy: 0.15, Current best_val: 0.4
count 249 / 600
iteration 0 / 20: loss 599.620912
count 250 / 600
iteration 0 / 20: loss 28.158254
count 251 / 600
iteration 0 / 20: loss 5.937797
iteration 0 / 1500: loss 5.884559
iteration 100 / 1500: loss 4.886805
iteration 200 / 1500: loss 4.441324
iteration 300 / 1500: loss 3.854054
iteration 400 / 1500: loss 4.166602
iteration 500 / 1500: loss 3.810284
iteration 600 / 1500: loss 3.531537
iteration 700 / 1500: loss 3.897161
iteration 800 / 1500: loss 3.431912
iteration 900 / 1500: loss 3.259447
iteration 1000 / 1500: loss 3.344473
iteration 1100 / 1500: loss 3.030527
iteration 1200 / 1500: loss 3.174121
iteration 1300 / 1500: loss 3.134582
iteration 1400 / 1500: loss 3.146871
Val_accuracy: 0.191, Current best_val: 0.4
count 252 / 600
iteration 0 / 20: loss 6.076721
iteration 0 / 1500: loss 6.477916
iteration 100 / 1500: loss 5.175845
iteration 200 / 1500: loss 5.758463
iteration 300 / 1500: loss 5.887576
iteration 400 / 1500: loss 5.364138
iteration 500 / 1500: loss 5.292358
iteration 600 / 1500: loss 5.372459
iteration 700 / 1500: loss 5.326791
iteration 800 / 1500: loss 5.314534
iteration 900 / 1500: loss 4.810876
iteration 1000 / 1500: loss 4.864037
iteration 1100 / 1500: loss 4.926878
iteration 1200 / 1500: loss 4.689074
iteration 1300 / 1500: loss 4.556022
iteration 1400 / 1500: loss 4.506254
Val_accuracy: 0.167, Current best_val: 0.4
count 253 / 600
iteration 0 / 20: loss 25.754111
count 254 / 600
iteration 0 / 20: loss 8.848588
iteration 0 / 1500: loss 9.154251
iteration 100 / 1500: loss 8.764648
iteration 200 / 1500: loss 8.362889
iteration 300 / 1500: loss 8.704518
iteration 400 / 1500: loss 8.673708
iteration 500 / 1500: loss 8.015803
iteration 600 / 1500: loss 8.557117
iteration 700 / 1500: loss 8.594715
iteration 800 / 1500: loss 8.778921
iteration 900 / 1500: loss 8.213088
iteration 1000 / 1500: loss 8.305694
iteration 1100 / 1500: loss 8.530635
iteration 1200 / 1500: loss 8.472539
iteration 1300 / 1500: loss 8.406215
iteration 1400 / 1500: loss 8.701394
Val_accuracy: 0.094, Current best_val: 0.4
count 255 / 600
iteration 0 / 20: loss 25.566197
count 256 / 600
iteration 0 / 20: loss 117.099835
count 257 / 600
iteration 0 / 20: loss 7.191440
iteration 0 / 1500: loss 7.626048
iteration 100 / 1500: loss 4.296229
iteration 200 / 1500: loss 3.772332
iteration 300 / 1500: loss 3.634189
iteration 400 / 1500: loss 3.514780
iteration 500 / 1500: loss 3.689317
iteration 600 / 1500: loss 3.341431
iteration 700 / 1500: loss 3.163637
iteration 800 / 1500: loss 3.506802
iteration 900 / 1500: loss 3.141053
iteration 1000 / 1500: loss 3.012519
iteration 1100 / 1500: loss 3.282474
iteration 1200 / 1500: loss 3.322832
iteration 1300 / 1500: loss 3.165926
iteration 1400 / 1500: loss 3.037166
Val_accuracy: 0.318, Current best_val: 0.4
count 258 / 600
iteration 0 / 20: loss 5.886367
iteration 0 / 1500: loss 5.739033
iteration 100 / 1500: loss 2.584850
iteration 200 / 1500: loss 2.381028
iteration 300 / 1500: loss 2.287799
iteration 400 / 1500: loss 2.141548
iteration 500 / 1500: loss 1.994945
iteration 600 / 1500: loss 2.136825
iteration 700 / 1500: loss 1.926900
iteration 800 / 1500: loss 1.962532
iteration 900 / 1500: loss 1.916983
iteration 1000 / 1500: loss 1.815221
iteration 1100 / 1500: loss 1.961028
iteration 1200 / 1500: loss 1.787233
iteration 1300 / 1500: loss 1.964773
iteration 1400 / 1500: loss 1.714468
Val_accuracy: 0.349, Current best_val: 0.4
count 259 / 600
iteration 0 / 20: loss 567.467986
count 260 / 600
iteration 0 / 20: loss 6.628150
iteration 0 / 1500: loss 6.900827
iteration 100 / 1500: loss 4.010294
iteration 200 / 1500: loss 3.911258
iteration 300 / 1500: loss 3.410173
iteration 400 / 1500: loss 3.622614
iteration 500 / 1500: loss 3.378633
iteration 600 / 1500: loss 3.352861
iteration 700 / 1500: loss 3.420173
iteration 800 / 1500: loss 3.431391
iteration 900 / 1500: loss 3.136909
iteration 1000 / 1500: loss 3.208978
iteration 1100 / 1500: loss 3.228447
iteration 1200 / 1500: loss 3.140284
iteration 1300 / 1500: loss 3.069084
iteration 1400 / 1500: loss 2.835782
Val_accuracy: 0.321, Current best_val: 0.4
count 261 / 600
iteration 0 / 20: loss 5.882391
iteration 0 / 1500: loss 6.190999
iteration 100 / 1500: loss 2.673362
iteration 200 / 1500: loss 2.417247
iteration 300 / 1500: loss 2.118658
iteration 400 / 1500: loss 2.070017
iteration 500 / 1500: loss 2.106951
iteration 600 / 1500: loss 1.869286
iteration 700 / 1500: loss 1.993617
iteration 800 / 1500: loss 1.817100
iteration 900 / 1500: loss 1.969231
iteration 1000 / 1500: loss 1.963685
iteration 1100 / 1500: loss 1.888381
iteration 1200 / 1500: loss 1.812600
iteration 1300 / 1500: loss 1.848414
iteration 1400 / 1500: loss 1.860628
Val_accuracy: 0.372, Current best_val: 0.4
count 262 / 600
iteration 0 / 20: loss 5.987752
iteration 0 / 1500: loss 5.204572
iteration 100 / 1500: loss 5.412523
iteration 200 / 1500: loss 5.688067
iteration 300 / 1500: loss 5.071442
iteration 400 / 1500: loss 5.099158
iteration 500 / 1500: loss 5.032498
iteration 600 / 1500: loss 4.593283
iteration 700 / 1500: loss 4.461443
iteration 800 / 1500: loss 4.710069
iteration 900 / 1500: loss 4.460329
iteration 1000 / 1500: loss 4.188250
iteration 1100 / 1500: loss 4.371691
iteration 1200 / 1500: loss 4.197094
iteration 1300 / 1500: loss 4.103748
iteration 1400 / 1500: loss 4.020892
Val_accuracy: 0.13, Current best_val: 0.4
count 263 / 600
iteration 0 / 20: loss 503.258381
count 264 / 600
iteration 0 / 20: loss 6.663209
iteration 0 / 1500: loss 4.804995
iteration 100 / 1500: loss 5.766185
iteration 200 / 1500: loss 5.075385
iteration 300 / 1500: loss 5.022012
iteration 400 / 1500: loss 5.216620
iteration 500 / 1500: loss 5.187682
iteration 600 / 1500: loss 4.971688
iteration 700 / 1500: loss 4.951478
iteration 800 / 1500: loss 5.181524
iteration 900 / 1500: loss 5.298555
iteration 1000 / 1500: loss 5.026826
iteration 1100 / 1500: loss 4.549783
iteration 1200 / 1500: loss 5.228412
iteration 1300 / 1500: loss 4.947520
iteration 1400 / 1500: loss 5.086043
Val_accuracy: 0.123, Current best_val: 0.4
count 265 / 600
iteration 0 / 20: loss 5.847572
iteration 0 / 1500: loss 5.586488
iteration 100 / 1500: loss 3.897432
iteration 200 / 1500: loss 3.712499
iteration 300 / 1500: loss 3.674119
iteration 400 / 1500: loss 3.385276
iteration 500 / 1500: loss 3.261301
iteration 600 / 1500: loss 3.264200
iteration 700 / 1500: loss 3.319149
iteration 800 / 1500: loss 3.056568
iteration 900 / 1500: loss 3.031159
iteration 1000 / 1500: loss 2.453533
iteration 1100 / 1500: loss 2.996736
iteration 1200 / 1500: loss 2.697507
iteration 1300 / 1500: loss 3.038727
iteration 1400 / 1500: loss 2.716047
Val_accuracy: 0.233, Current best_val: 0.4
count 266 / 600
iteration 0 / 20: loss 5.864868
iteration 0 / 1500: loss 5.259598
iteration 100 / 1500: loss 5.386410
iteration 200 / 1500: loss 5.285496
iteration 300 / 1500: loss 4.907678
iteration 400 / 1500: loss 5.075747
iteration 500 / 1500: loss 4.833801
iteration 600 / 1500: loss 4.761143
iteration 700 / 1500: loss 4.582227
iteration 800 / 1500: loss 5.053757
iteration 900 / 1500: loss 4.392280
iteration 1000 / 1500: loss 4.297381
iteration 1100 / 1500: loss 4.691639
iteration 1200 / 1500: loss 4.748000
iteration 1300 / 1500: loss 4.333032
iteration 1400 / 1500: loss 4.161574
Val_accuracy: 0.171, Current best_val: 0.4
count 267 / 600
iteration 0 / 20: loss 3076.744161
count 268 / 600
iteration 0 / 20: loss 344.344552
count 269 / 600
iteration 0 / 20: loss 37.670617
count 270 / 600
iteration 0 / 20: loss 6.045535
iteration 0 / 1500: loss 5.732254
iteration 100 / 1500: loss 4.436955
iteration 200 / 1500: loss 3.984460
iteration 300 / 1500: loss 3.865115
iteration 400 / 1500: loss 3.325783
iteration 500 / 1500: loss 3.357129
iteration 600 / 1500: loss 3.320336
iteration 700 / 1500: loss 2.928021
iteration 800 / 1500: loss 3.024193
iteration 900 / 1500: loss 3.242858
iteration 1000 / 1500: loss 3.319040
iteration 1100 / 1500: loss 3.422105
iteration 1200 / 1500: loss 2.880826
iteration 1300 / 1500: loss 3.132823
iteration 1400 / 1500: loss 3.141217
Val_accuracy: 0.239, Current best_val: 0.4
count 271 / 600
iteration 0 / 20: loss 102.363384
count 272 / 600
iteration 0 / 20: loss 273.942496
count 273 / 600
iteration 0 / 20: loss 6.267778
iteration 0 / 1500: loss 5.134763
iteration 100 / 1500: loss 2.443694
iteration 200 / 1500: loss 2.212494
iteration 300 / 1500: loss 2.017417
iteration 400 / 1500: loss 2.231184
iteration 500 / 1500: loss 1.965219
iteration 600 / 1500: loss 2.096634
iteration 700 / 1500: loss 1.999924
iteration 800 / 1500: loss 1.854626
iteration 900 / 1500: loss 1.902315
iteration 1000 / 1500: loss 1.868736
iteration 1100 / 1500: loss 1.853028
iteration 1200 / 1500: loss 1.950697
iteration 1300 / 1500: loss 1.749537
iteration 1400 / 1500: loss 1.851690
Val_accuracy: 0.345, Current best_val: 0.4
count 274 / 600
iteration 0 / 20: loss 17.296549
count 275 / 600
iteration 0 / 20: loss 6.184061
iteration 0 / 1500: loss 6.126685
iteration 100 / 1500: loss 5.687137
iteration 200 / 1500: loss 5.209050
iteration 300 / 1500: loss 5.694745
iteration 400 / 1500: loss 4.906421
iteration 500 / 1500: loss 5.565167
iteration 600 / 1500: loss 5.018955
iteration 700 / 1500: loss 5.039134
iteration 800 / 1500: loss 4.920545
iteration 900 / 1500: loss 4.802510
iteration 1000 / 1500: loss 5.457199
iteration 1100 / 1500: loss 5.170177
iteration 1200 / 1500: loss 4.958516
iteration 1300 / 1500: loss 4.633907
iteration 1400 / 1500: loss 5.225833
Val_accuracy: 0.114, Current best_val: 0.4
count 276 / 600
iteration 0 / 20: loss 225.520660
count 277 / 600
iteration 0 / 20: loss 5.390280
iteration 0 / 1500: loss 6.242005
iteration 100 / 1500: loss 2.982137
iteration 200 / 1500: loss 2.810751
iteration 300 / 1500: loss 2.584532
iteration 400 / 1500: loss 2.464241
iteration 500 / 1500: loss 2.419067
iteration 600 / 1500: loss 2.216856
iteration 700 / 1500: loss 2.139978
iteration 800 / 1500: loss 2.307809
iteration 900 / 1500: loss 2.279008
iteration 1000 / 1500: loss 2.116403
iteration 1100 / 1500: loss 2.072199
iteration 1200 / 1500: loss 2.339112
iteration 1300 / 1500: loss 1.997435
iteration 1400 / 1500: loss 2.380820
Val_accuracy: 0.323, Current best_val: 0.4
count 278 / 600
iteration 0 / 20: loss 8.746894
iteration 0 / 1500: loss 9.717092
iteration 100 / 1500: loss 9.136844
iteration 200 / 1500: loss 9.037557
iteration 300 / 1500: loss 9.040448
iteration 400 / 1500: loss 8.370415
iteration 500 / 1500: loss 8.353043
iteration 600 / 1500: loss 8.242117
iteration 700 / 1500: loss 8.364185
iteration 800 / 1500: loss 7.882126
iteration 900 / 1500: loss 8.225385
iteration 1000 / 1500: loss 7.847908
iteration 1100 / 1500: loss 7.507799
iteration 1200 / 1500: loss 7.518904
iteration 1300 / 1500: loss 7.674933
iteration 1400 / 1500: loss 7.624660
Val_accuracy: 0.136, Current best_val: 0.4
count 279 / 600
```

```
iteration 0 / 20: loss 5.453895
iteration 0 / 1500: loss 5.385933
iteration 100 / 1500: loss 5.360792
iteration 200 / 1500: loss 4.994010
iteration 300 / 1500: loss 4.286447
iteration 400 / 1500: loss 5.009115
iteration 500 / 1500: loss 4.602746
iteration 600 / 1500: loss 4.588302
iteration 700 / 1500: loss 4.918763
iteration 800 / 1500: loss 4.609540
iteration 900 / 1500: loss 4.174616
iteration 1000 / 1500: loss 3.837461
iteration 1100 / 1500: loss 4.036033
iteration 1200 / 1500: loss 3.780355
iteration 1300 / 1500: loss 3.839009
iteration 1400 / 1500: loss 3.632733
Val_accuracy: 0.188, Current best_val: 0.4
count 280 / 600
iteration 0 / 20: loss 1403.061366
count 281 / 600
iteration 0 / 20: loss 5.479934
iteration 0 / 1500: loss 5.871369
iteration 100 / 1500: loss 5.530366
iteration 200 / 1500: loss 4.402155
iteration 300 / 1500: loss 4.930190
iteration 400 / 1500: loss 4.365491
iteration 500 / 1500: loss 4.195068
iteration 600 / 1500: loss 4.480570
iteration 700 / 1500: loss 4.269778
iteration 800 / 1500: loss 4.149114
iteration 900 / 1500: loss 4.358005
iteration 1000 / 1500: loss 3.873490
iteration 1100 / 1500: loss 4.046837
iteration 1200 / 1500: loss 3.711381
iteration 1300 / 1500: loss 3.766645
iteration 1400 / 1500: loss 3.802813
Val_accuracy: 0.184, Current best_val: 0.4
count 282 / 600
iteration 0 / 20: loss 5.503102
iteration 0 / 1500: loss 5.382868
iteration 100 / 1500: loss 3.817049
iteration 200 / 1500: loss 3.342255
iteration 300 / 1500: loss 3.343474
iteration 400 / 1500: loss 3.081914
iteration 500 / 1500: loss 3.029656
iteration 600 / 1500: loss 3.109833
iteration 700 / 1500: loss 2.774487
iteration 800 / 1500: loss 2.673761
iteration 900 / 1500: loss 2.687232
iteration 1000 / 1500: loss 2.516894
iteration 1100 / 1500: loss 2.647411
iteration 1200 / 1500: loss 2.647969
iteration 1300 / 1500: loss 2.502684
iteration 1400 / 1500: loss 2.485214
Val_accuracy: 0.284, Current best_val: 0.4
count 283 / 600
iteration 0 / 20: loss 36.739391
count 284 / 600
iteration 0 / 20: loss 681.273066
count 285 / 600
iteration 0 / 20: loss 25.979089
count 286 / 600
iteration 0 / 20: loss 138.233657
count 287 / 600
iteration 0 / 20: loss 14.351782
iteration 0 / 1500: loss 13.106228
iteration 100 / 1500: loss 11.015496
iteration 200 / 1500: loss 10.128835
iteration 300 / 1500: loss 10.209984
iteration 400 / 1500: loss 9.580848
iteration 500 / 1500: loss 9.162461
iteration 600 / 1500: loss 8.770969
iteration 700 / 1500: loss 8.719657
iteration 800 / 1500: loss 8.315365
iteration 900 / 1500: loss 8.290874
iteration 1000 / 1500: loss 7.629676
iteration 1100 / 1500: loss 7.443312
iteration 1200 / 1500: loss 7.097530
iteration 1300 / 1500: loss 7.011625
iteration 1400 / 1500: loss 6.777809
Val_accuracy: 0.3, Current best_val: 0.4
count 288 / 600
iteration 0 / 20: loss 5.539575
iteration 0 / 1500: loss 6.463831
iteration 100 / 1500: loss 2.607504
iteration 200 / 1500: loss 2.378119
iteration 300 / 1500: loss 2.388970
iteration 400 / 1500: loss 2.230295
iteration 500 / 1500: loss 2.110666
iteration 600 / 1500: loss 2.018627
iteration 700 / 1500: loss 1.996954
iteration 800 / 1500: loss 1.903043
iteration 900 / 1500: loss 1.892110
iteration 1000 / 1500: loss 2.108020
iteration 1100 / 1500: loss 2.020862
iteration 1200 / 1500: loss 2.057345
iteration 1300 / 1500: loss 1.855711
iteration 1400 / 1500: loss 2.002227
Val_accuracy: 0.355, Current best_val: 0.4
count 289 / 600
iteration 0 / 20: loss 286.500538
count 290 / 600
iteration 0 / 20: loss 5.065619
iteration 0 / 1500: loss 5.993063
iteration 100 / 1500: loss 5.180701
iteration 200 / 1500: loss 5.666724
iteration 300 / 1500: loss 5.515668
iteration 400 / 1500: loss 5.711769
iteration 500 / 1500: loss 6.410260
iteration 600 / 1500: loss 5.856438
iteration 700 / 1500: loss 5.211530
iteration 800 / 1500: loss 5.385729
iteration 900 / 1500: loss 5.560155
iteration 1000 / 1500: loss 5.366603
iteration 1100 / 1500: loss 5.490949
iteration 1200 / 1500: loss 4.964260
iteration 1300 / 1500: loss 4.844881
iteration 1400 / 1500: loss 4.961938
Val_accuracy: 0.103, Current best_val: 0.4
count 291 / 600
iteration 0 / 20: loss 5.093958
iteration 0 / 1500: loss 6.413428
iteration 100 / 1500: loss 7.073510
iteration 200 / 1500: loss 5.611568
iteration 300 / 1500: loss 5.230692
iteration 400 / 1500: loss 4.811873
iteration 500 / 1500: loss 4.703755
iteration 600 / 1500: loss 4.788581
iteration 700 / 1500: loss 4.678130
iteration 800 / 1500: loss 4.546870
iteration 900 / 1500: loss 4.185850
iteration 1000 / 1500: loss 4.377335
iteration 1100 / 1500: loss 4.134409
iteration 1200 / 1500: loss 4.197917
iteration 1300 / 1500: loss 3.961570
iteration 1400 / 1500: loss 3.727967
Val_accuracy: 0.155, Current best_val: 0.4
count 292 / 600
iteration 0 / 20: loss 6.058747
iteration 0 / 1500: loss 4.863190
iteration 100 / 1500: loss 4.899464
iteration 200 / 1500: loss 4.592483
iteration 300 / 1500: loss 4.544968
iteration 400 / 1500: loss 5.045419
iteration 500 / 1500: loss 5.163295
iteration 600 / 1500: loss 4.659638
iteration 700 / 1500: loss 5.071026
iteration 800 / 1500: loss 4.795319
iteration 900 / 1500: loss 4.872400
iteration 1000 / 1500: loss 4.607665
iteration 1100 / 1500: loss 4.707351
iteration 1200 / 1500: loss 5.015650
iteration 1300 / 1500: loss 4.326690
iteration 1400 / 1500: loss 4.947422
Val_accuracy: 0.125, Current best_val: 0.4
count 293 / 600
iteration 0 / 20: loss 50.404518
count 294 / 600
iteration 0 / 20: loss 5.789491
iteration 0 / 1500: loss 5.772321
iteration 100 / 1500: loss 3.939527
iteration 200 / 1500: loss 3.418271
iteration 300 / 1500: loss 3.429231
iteration 400 / 1500: loss 3.180761
iteration 500 / 1500: loss 3.065014
iteration 600 / 1500: loss 3.180461
iteration 700 / 1500: loss 2.735843
iteration 800 / 1500: loss 2.701786
iteration 900 / 1500: loss 2.762797
iteration 1000 / 1500: loss 2.705156
iteration 1100 / 1500: loss 2.545858
iteration 1200 / 1500: loss 2.690880
iteration 1300 / 1500: loss 2.514877
iteration 1400 / 1500: loss 2.641210
Val_accuracy: 0.251, Current best_val: 0.4
count 295 / 600
iteration 0 / 20: loss 2196.544430
count 296 / 600
iteration 0 / 20: loss 6.330441
iteration 0 / 1500: loss 6.191883
iteration 100 / 1500: loss 5.662818
iteration 200 / 1500: loss 5.317405
iteration 300 / 1500: loss 5.617861
iteration 400 / 1500: loss 5.101085
iteration 500 / 1500: loss 5.248169
iteration 600 / 1500: loss 4.884671
iteration 700 / 1500: loss 4.813595
iteration 800 / 1500: loss 5.332566
iteration 900 / 1500: loss 4.745628
iteration 1000 / 1500: loss 4.672314
iteration 1100 / 1500: loss 4.092187
iteration 1200 / 1500: loss 4.670952
iteration 1300 / 1500: loss 5.163831
iteration 1400 / 1500: loss 4.215832
Val_accuracy: 0.122, Current best_val: 0.4
count 297 / 600
iteration 0 / 20: loss 4.978178
iteration 0 / 1500: loss 5.934369
iteration 100 / 1500: loss 2.344799
iteration 200 / 1500: loss 2.233574
iteration 300 / 1500: loss 2.177565
iteration 400 / 1500: loss 2.013029
iteration 500 / 1500: loss 1.798396
iteration 600 / 1500: loss 1.874950
iteration 700 / 1500: loss 2.001260
iteration 800 / 1500: loss 1.974746
iteration 900 / 1500: loss 2.062150
iteration 1000 / 1500: loss 1.973605
iteration 1100 / 1500: loss 1.886310
iteration 1200 / 1500: loss 1.780200
iteration 1300 / 1500: loss 1.848937
iteration 1400 / 1500: loss 1.753052
Val_accuracy: 0.372, Current best_val: 0.4
count 298 / 600
iteration 0 / 20: loss 201.588514
count 299 / 600
iteration 0 / 20: loss 428.199776
count 300 / 600
iteration 0 / 20: loss 5.128568
iteration 0 / 1500: loss 5.061981
iteration 100 / 1500: loss 5.163039
iteration 200 / 1500: loss 4.887293
iteration 300 / 1500: loss 4.791766
iteration 400 / 1500: loss 4.882720
iteration 500 / 1500: loss 4.412925
iteration 600 / 1500: loss 4.339180
iteration 700 / 1500: loss 4.638335
iteration 800 / 1500: loss 4.982029
iteration 900 / 1500: loss 4.178960
iteration 1000 / 1500: loss 4.558793
iteration 1100 / 1500: loss 4.877886
iteration 1200 / 1500: loss 4.304125
iteration 1300 / 1500: loss 4.606068
iteration 1400 / 1500: loss 4.738237
Val_accuracy: 0.139, Current best_val: 0.4
count 301 / 600
iteration 0 / 20: loss 10.525192
iteration 0 / 1500: loss 10.533753
iteration 100 / 1500: loss 5.581194
iteration 200 / 1500: loss 4.358886
iteration 300 / 1500: loss 3.678555
iteration 400 / 1500: loss 3.089586
iteration 500 / 1500: loss 2.581674
iteration 600 / 1500: loss 2.427969
iteration 700 / 1500: loss 2.255471
iteration 800 / 1500: loss 2.105578
iteration 900 / 1500: loss 1.995903
iteration 1000 / 1500: loss 1.985881
iteration 1100 / 1500: loss 1.953263
iteration 1200 / 1500: loss 1.892744
iteration 1300 / 1500: loss 1.951299
iteration 1400 / 1500: loss 1.740618
Val_accuracy: 0.364, Current best_val: 0.4
count 302 / 600
iteration 0 / 20: loss 85.394223
count 303 / 600
iteration 0 / 20: loss 5.436743
iteration 0 / 1500: loss 4.939883
iteration 100 / 1500: loss 5.177191
iteration 200 / 1500: loss 5.370016
iteration 300 / 1500: loss 4.883999
iteration 400 / 1500: loss 4.568874
iteration 500 / 1500: loss 5.172614
iteration 600 / 1500: loss 4.758274
iteration 700 / 1500: loss 4.887453
iteration 800 / 1500: loss 4.873752
iteration 900 / 1500: loss 4.871563
iteration 1000 / 1500: loss 4.815590
iteration 1100 / 1500: loss 5.076773
iteration 1200 / 1500: loss 4.993664
iteration 1300 / 1500: loss 4.810045
iteration 1400 / 1500: loss 4.593041
Val_accuracy: 0.119, Current best_val: 0.4
count 304 / 600
iteration 0 / 20: loss 6.390007
iteration 0 / 1500: loss 6.403840
iteration 100 / 1500: loss 6.536910
iteration 200 / 1500: loss 5.806877
iteration 300 / 1500: loss 6.252279
iteration 400 / 1500: loss 5.570668
iteration 500 / 1500: loss 5.850288
iteration 600 / 1500: loss 5.501427
iteration 700 / 1500: loss 5.887050
iteration 800 / 1500: loss 5.693701
iteration 900 / 1500: loss 5.260412
iteration 1000 / 1500: loss 5.452476
iteration 1100 / 1500: loss 5.306146
iteration 1200 / 1500: loss 5.402400
iteration 1300 / 1500: loss 5.072635
iteration 1400 / 1500: loss 5.452749
Val_accuracy: 0.145, Current best_val: 0.4
count 305 / 600
iteration 0 / 20: loss 6.129360
iteration 0 / 1500: loss 5.492336
iteration 100 / 1500: loss 2.927842
iteration 200 / 1500: loss 2.253570
iteration 300 / 1500: loss 2.315774
iteration 400 / 1500: loss 2.389475
iteration 500 / 1500: loss 2.550810
iteration 600 / 1500: loss 2.311244
iteration 700 / 1500: loss 2.374735
iteration 800 / 1500: loss 2.219572
iteration 900 / 1500: loss 2.102628
iteration 1000 / 1500: loss 2.145080
iteration 1100 / 1500: loss 2.199298
iteration 1200 / 1500: loss 2.080001
iteration 1300 / 1500: loss 2.251807
iteration 1400 / 1500: loss 2.006031
Val_accuracy: 0.33, Current best_val: 0.4
count 306 / 600
iteration 0 / 20: loss 6.066247
iteration 0 / 1500: loss 6.116801
iteration 100 / 1500: loss 3.204868
iteration 200 / 1500: loss 2.938645
iteration 300 / 1500: loss 2.688033
iteration 400 / 1500: loss 2.316091
iteration 500 / 1500: loss 2.365514
iteration 600 / 1500: loss 2.630414
iteration 700 / 1500: loss 2.530460
iteration 800 / 1500: loss 2.549378
iteration 900 / 1500: loss 2.345597
iteration 1000 / 1500: loss 2.251372
iteration 1100 / 1500: loss 2.296171
iteration 1200 / 1500: loss 2.355763
iteration 1300 / 1500: loss 2.251147
iteration 1400 / 1500: loss 2.219755
Val_accuracy: 0.308, Current best_val: 0.4
count 307 / 600
iteration 0 / 20: loss 33.851516
count 308 / 600
iteration 0 / 20: loss 7.219888
iteration 0 / 1500: loss 6.474563
iteration 100 / 1500: loss 5.836521
iteration 200 / 1500: loss 5.443626
iteration 300 / 1500: loss 4.954898
iteration 400 / 1500: loss 5.231636
iteration 500 / 1500: loss 4.890176
iteration 600 / 1500: loss 4.983199
iteration 700 / 1500: loss 4.460259
iteration 800 / 1500: loss 4.105586
iteration 900 / 1500: loss 4.128151
iteration 1000 / 1500: loss 4.305551
iteration 1100 / 1500: loss 3.983812
iteration 1200 / 1500: loss 4.100771
iteration 1300 / 1500: loss 4.234609
iteration 1400 / 1500: loss 4.019754
Val_accuracy: 0.204, Current best_val: 0.4
count 309 / 600
iteration 0 / 20: loss 1328.413786
count 310 / 600
iteration 0 / 20: loss 5.583578
iteration 0 / 1500: loss 7.179618
iteration 100 / 1500: loss 6.735957
iteration 200 / 1500: loss 5.959841
iteration 300 / 1500: loss 5.970261
iteration 400 / 1500: loss 5.617394
iteration 500 / 1500: loss 5.791855
iteration 600 / 1500: loss 5.568538
iteration 700 / 1500: loss 5.183688
iteration 800 / 1500: loss 5.099547
iteration 900 / 1500: loss 5.002806
iteration 1000 / 1500: loss 5.070892
iteration 1100 / 1500: loss 4.865042
iteration 1200 / 1500: loss 4.858037
iteration 1300 / 1500: loss 4.802917
iteration 1400 / 1500: loss 4.407928
Val_accuracy: 0.14, Current best_val: 0.4
count 311 / 600
iteration 0 / 20: loss 129.954779
count 312 / 600
iteration 0 / 20: loss 1281.572901
count 313 / 600
iteration 0 / 20: loss 6.401626
iteration 0 / 1500: loss 6.795898
iteration 100 / 1500: loss 2.499339
iteration 200 / 1500: loss 2.150531
iteration 300 / 1500: loss 2.245348
iteration 400 / 1500: loss 1.965450
iteration 500 / 1500: loss 2.133026
iteration 600 / 1500: loss 1.991057
iteration 700 / 1500: loss 2.013644
iteration 800 / 1500: loss 1.822140
iteration 900 / 1500: loss 1.881732
iteration 1000 / 1500: loss 1.990088
iteration 1100 / 1500: loss 1.971511
iteration 1200 / 1500: loss 1.914211
iteration 1300 / 1500: loss 1.912414
iteration 1400 / 1500: loss 1.727218
Val_accuracy: 0.367, Current best_val: 0.4
count 314 / 600
iteration 0 / 20: loss 5.096504
iteration 0 / 1500: loss 5.880075
iteration 100 / 1500: loss 4.383384
iteration 200 / 1500: loss 4.269061
iteration 300 / 1500: loss 3.584747
iteration 400 / 1500: loss 3.692028
iteration 500 / 1500: loss 3.195385
iteration 600 / 1500: loss 3.392033
iteration 700 / 1500: loss 3.025425
iteration 800 / 1500: loss 3.091032
iteration 900 / 1500: loss 3.215804
iteration 1000 / 1500: loss 3.236981
iteration 1100 / 1500: loss 2.968272
iteration 1200 / 1500: loss 2.895839
iteration 1300 / 1500: loss 3.106627
iteration 1400 / 1500: loss 2.791417
Val_accuracy: 0.234, Current best_val: 0.4
count 315 / 600
iteration 0 / 20: loss 165.763469
count 316 / 600
iteration 0 / 20: loss 10.803632
iteration 0 / 1500: loss 10.491517
iteration 100 / 1500: loss 9.504981
iteration 200 / 1500: loss 8.911614
iteration 300 / 1500: loss 8.174029
iteration 400 / 1500: loss 8.651530
iteration 500 / 1500: loss 8.393761
iteration 600 / 1500: loss 8.049866
iteration 700 / 1500: loss 8.084610
iteration 800 / 1500: loss 7.647672
iteration 900 / 1500: loss 7.749824
iteration 1000 / 1500: loss 7.991671
iteration 1100 / 1500: loss 7.820730
iteration 1200 / 1500: loss 7.716972
iteration 1300 / 1500: loss 7.701190
iteration 1400 / 1500: loss 7.530393
Val_accuracy: 0.198, Current best_val: 0.4
count 317 / 600
iteration 0 / 20: loss 6.554087
iteration 0 / 1500: loss 6.328501
iteration 100 / 1500: loss 4.616997
iteration 200 / 1500: loss 3.930135
iteration 300 / 1500: loss 3.603468
iteration 400 / 1500: loss 3.459457
iteration 500 / 1500: loss 3.365955
iteration 600 / 1500: loss 3.113244
iteration 700 / 1500: loss 2.973879
iteration 800 / 1500: loss 3.038762
iteration 900 / 1500: loss 3.117203
iteration 1000 / 1500: loss 2.960421
iteration 1100 / 1500: loss 3.213448
iteration 1200 / 1500: loss 3.128128
iteration 1300 / 1500: loss 3.161153
iteration 1400 / 1500: loss 2.850959
Val_accuracy: 0.256, Current best_val: 0.4
count 318 / 600
iteration 0 / 20: loss 14.288851
count 319 / 600
iteration 0 / 20: loss 9.559765
iteration 0 / 1500: loss 9.261075
iteration 100 / 1500: loss 4.860897
iteration 200 / 1500: loss 3.862257
iteration 300 / 1500: loss 3.293030
iteration 400 / 1500: loss 3.106172
iteration 500 / 1500: loss 2.620723
iteration 600 / 1500: loss 2.461468
iteration 700 / 1500: loss 2.383162
iteration 800 / 1500: loss 2.348039
iteration 900 / 1500: loss 2.049300
iteration 1000 / 1500: loss 2.103686
iteration 1100 / 1500: loss 2.132822
iteration 1200 / 1500: loss 2.009102
iteration 1300 / 1500: loss 2.082057
iteration 1400 / 1500: loss 2.300149
Val_accuracy: 0.393, Current best_val: 0.4
count 320 / 600
iteration 0 / 20: loss 1589.040555
count 321 / 600
iteration 0 / 20: loss 5.522917
iteration 0 / 1500: loss 4.926405
iteration 100 / 1500: loss 5.197128
iteration 200 / 1500: loss 5.702908
iteration 300 / 1500: loss 5.557689
iteration 400 / 1500: loss 4.589745
iteration 500 / 1500: loss 5.383288
iteration 600 / 1500: loss 5.017945
iteration 700 / 1500: loss 4.877616
iteration 800 / 1500: loss 4.779189
iteration 900 / 1500: loss 4.722846
iteration 1000 / 1500: loss 4.923730
iteration 1100 / 1500: loss 4.458859
iteration 1200 / 1500: loss 5.008220
iteration 1300 / 1500: loss 4.881189
iteration 1400 / 1500: loss 4.232362
Val_accuracy: 0.131, Current best_val: 0.4
count 322 / 600
iteration 0 / 20: loss 32.836097
count 323 / 600
iteration 0 / 20: loss 5.116847
iteration 0 / 1500: loss 5.787758
iteration 100 / 1500: loss 5.840671
iteration 200 / 1500: loss 5.576238
iteration 300 / 1500: loss 5.070896
iteration 400 / 1500: loss 5.957164
iteration 500 / 1500: loss 5.538839
iteration 600 / 1500: loss 5.117638
iteration 700 / 1500: loss 5.482383
iteration 800 / 1500: loss 5.176621
iteration 900 / 1500: loss 5.049583
iteration 1000 / 1500: loss 5.121082
iteration 1100 / 1500: loss 5.567470
iteration 1200 / 1500: loss 4.959722
iteration 1300 / 1500: loss 5.147081
iteration 1400 / 1500: loss 4.643034
Val_accuracy: 0.135, Current best_val: 0.4
count 324 / 600
iteration 0 / 20: loss 7.248574
iteration 0 / 1500: loss 6.578706
iteration 100 / 1500: loss 3.098565
iteration 200 / 1500: loss 2.804310
iteration 300 / 1500: loss 3.145338
iteration 400 / 1500: loss 2.656984
iteration 500 / 1500: loss 2.659124
iteration 600 / 1500: loss 2.846994
iteration 700 / 1500: loss 2.883216
iteration 800 / 1500: loss 2.396886
iteration 900 / 1500: loss 2.326547
iteration 1000 / 1500: loss 2.472472
iteration 1100 / 1500: loss 2.632722
iteration 1200 / 1500: loss 3.220756
iteration 1300 / 1500: loss 3.019585
iteration 1400 / 1500: loss 2.114293
Val_accuracy: 0.333, Current best_val: 0.4
count 325 / 600
iteration 0 / 20: loss 5.352145
iteration 0 / 1500: loss 5.009957
iteration 100 / 1500: loss 2.602241
iteration 200 / 1500: loss 2.402455
iteration 300 / 1500: loss 2.354663
iteration 400 / 1500: loss 2.205410
iteration 500 / 1500: loss 1.952775
iteration 600 / 1500: loss 2.116372
iteration 700 / 1500: loss 2.135559
iteration 800 / 1500: loss 1.929492
iteration 900 / 1500: loss 2.017382
iteration 1000 / 1500: loss 2.059511
iteration 1100 / 1500: loss 1.946593
iteration 1200 / 1500: loss 1.981193
iteration 1300 / 1500: loss 1.920970
iteration 1400 / 1500: loss 1.908500
Val_accuracy: 0.362, Current best_val: 0.4
count 326 / 600
iteration 0 / 20: loss 7.439028
iteration 0 / 1500: loss 7.876461
iteration 100 / 1500: loss 7.581124
iteration 200 / 1500: loss 7.050682
iteration 300 / 1500: loss 7.571467
iteration 400 / 1500: loss 7.304613
iteration 500 / 1500: loss 7.034847
iteration 600 / 1500: loss 6.934840
iteration 700 / 1500: loss 6.771401
iteration 800 / 1500: loss 6.932418
iteration 900 / 1500: loss 7.087733
iteration 1000 / 1500: loss 6.179011
iteration 1100 / 1500: loss 6.378914
iteration 1200 / 1500: loss 6.360875
iteration 1300 / 1500: loss 6.593374
iteration 1400 / 1500: loss 6.370194
Val_accuracy: 0.126, Current best_val: 0.4
count 327 / 600
iteration 0 / 20: loss 9.195197
iteration 0 / 1500: loss 10.019786
iteration 100 / 1500: loss 9.423418
iteration 200 / 1500: loss 8.637312
iteration 300 / 1500: loss 8.556393
iteration 400 / 1500: loss 8.266057
iteration 500 / 1500: loss 8.149694
iteration 600 / 1500: loss 7.971764
iteration 700 / 1500: loss 8.150028
iteration 800 / 1500: loss 7.791167
iteration 900 / 1500: loss 7.705082
iteration 1000 / 1500: loss 7.753842
iteration 1100 / 1500: loss 7.606902
iteration 1200 / 1500: loss 7.290054
iteration 1300 / 1500: loss 7.427035
iteration 1400 / 1500: loss 7.314843
```

```
Val_accuracy: 0.176, Current best_val: 0.4
count 328 / 600
iteration 0 / 20: loss 7.157157
iteration 0 / 1500: loss 5.218673
iteration 100 / 1500: loss 3.484985
iteration 200 / 1500: loss 3.310505
iteration 300 / 1500: loss 2.937422
iteration 400 / 1500: loss 2.808099
iteration 500 / 1500: loss 2.788310
iteration 600 / 1500: loss 2.828843
iteration 700 / 1500: loss 2.614419
iteration 800 / 1500: loss 2.653302
iteration 900 / 1500: loss 2.707564
iteration 1000 / 1500: loss 2.467954
iteration 1100 / 1500: loss 2.458097
iteration 1200 / 1500: loss 2.482699
iteration 1300 / 1500: loss 2.348605
iteration 1400 / 1500: loss 2.422631
Val_accuracy: 0.322, Current best_val: 0.4
count 329 / 600
iteration 0 / 20: loss 131.087088
count 330 / 600
iteration 0 / 20: loss 7.781843
iteration 0 / 1500: loss 7.953082
iteration 100 / 1500: loss 5.057057
iteration 200 / 1500: loss 4.722805
iteration 300 / 1500: loss 4.497762
iteration 400 / 1500: loss 4.479899
iteration 500 / 1500: loss 4.121886
iteration 600 / 1500: loss 4.044673
iteration 700 / 1500: loss 4.157569
iteration 800 / 1500: loss 4.071803
iteration 900 / 1500: loss 4.083471
iteration 1000 / 1500: loss 4.157611
iteration 1100 / 1500: loss 3.697821
iteration 1200 / 1500: loss 3.699592
iteration 1300 / 1500: loss 3.608762
iteration 1400 / 1500: loss 3.665142
Val_accuracy: 0.334, Current best_val: 0.4
count 331 / 600
iteration 0 / 20: loss 7.354716
iteration 0 / 1500: loss 5.118986
iteration 100 / 1500: loss 5.103321
iteration 200 / 1500: loss 4.847488
iteration 300 / 1500: loss 4.520121
iteration 400 / 1500: loss 4.407099
iteration 500 / 1500: loss 4.042621
iteration 600 / 1500: loss 4.154888
iteration 700 / 1500: loss 3.943277
iteration 800 / 1500: loss 3.654924
iteration 900 / 1500: loss 4.139843
iteration 1000 / 1500: loss 3.903755
iteration 1100 / 1500: loss 3.746733
iteration 1200 / 1500: loss 3.561012
iteration 1300 / 1500: loss 3.788738
iteration 1400 / 1500: loss 3.624825
Val_accuracy: 0.172, Current best_val: 0.4
count 332 / 600
iteration 0 / 20: loss 18.987491
iteration 0 / 1500: loss 17.986961
iteration 100 / 1500: loss 7.383700
iteration 200 / 1500: loss 3.141887
iteration 300 / 1500: loss 3.427438
iteration 400 / 1500: loss 2.409432
iteration 500 / 1500: loss 2.686816
iteration 600 / 1500: loss 2.264446
iteration 700 / 1500: loss 2.136187
iteration 800 / 1500: loss 2.314985
iteration 900 / 1500: loss 2.774211
iteration 1000 / 1500: loss 2.092616
iteration 1100 / 1500: loss 2.370572
iteration 1200 / 1500: loss 3.335086
iteration 1300 / 1500: loss 2.676594
iteration 1400 / 1500: loss 3.388707
Val_accuracy: 0.305, Current best_val: 0.4
count 333 / 600
iteration 0 / 20: loss 62.451929
count 334 / 600
iteration 0 / 20: loss 186.850941
count 335 / 600
iteration 0 / 20: loss 16.702006
count 336 / 600
iteration 0 / 20: loss 5.396820
iteration 0 / 1500: loss 5.439602
iteration 100 / 1500: loss 5.377179
iteration 200 / 1500: loss 5.193120
iteration 300 / 1500: loss 5.347656
iteration 400 / 1500: loss 5.366788
iteration 500 / 1500: loss 5.408964
iteration 600 / 1500: loss 4.935120
iteration 700 / 1500: loss 5.086484
iteration 800 / 1500: loss 4.936366
iteration 900 / 1500: loss 4.930906
iteration 1000 / 1500: loss 4.722940
iteration 1100 / 1500: loss 4.469919
iteration 1200 / 1500: loss 4.907491
iteration 1300 / 1500: loss 4.798159
iteration 1400 / 1500: loss 4.716666
Val_accuracy: 0.136, Current best_val: 0.4
count 337 / 600
iteration 0 / 20: loss 10.082304
iteration 0 / 1500: loss 10.091849
iteration 100 / 1500: loss 4.758548
iteration 200 / 1500: loss 4.340634
iteration 300 / 1500: loss 3.384470
iteration 400 / 1500: loss 3.149903
iteration 500 / 1500: loss 2.715847
iteration 600 / 1500: loss 2.749334
iteration 700 / 1500: loss 2.348757
iteration 800 / 1500: loss 2.425315
iteration 900 / 1500: loss 2.305529
iteration 1000 / 1500: loss 2.378575
iteration 1100 / 1500: loss 2.963590
iteration 1200 / 1500: loss 1.957276
iteration 1300 / 1500: loss 2.194752
iteration 1400 / 1500: loss 2.150084
Val_accuracy: 0.303, Current best_val: 0.4
count 338 / 600
iteration 0 / 20: loss 12.536166
iteration 0 / 1500: loss 11.878541
iteration 100 / 1500: loss 8.994825
iteration 200 / 1500: loss 7.828137
iteration 300 / 1500: loss 7.156993
iteration 400 / 1500: loss 6.306636
iteration 500 / 1500: loss 5.872926
iteration 600 / 1500: loss 5.536419
iteration 700 / 1500: loss 5.050826
iteration 800 / 1500: loss 4.734192
iteration 900 / 1500: loss 4.420275
iteration 1000 / 1500: loss 4.066374
iteration 1100 / 1500: loss 3.946374
iteration 1200 / 1500: loss 3.625676
iteration 1300 / 1500: loss 3.505926
iteration 1400 / 1500: loss 3.267424
Val_accuracy: 0.381, Current best_val: 0.4
count 339 / 600
iteration 0 / 20: loss 59.004132
count 340 / 600
iteration 0 / 20: loss 5.043162
iteration 0 / 1500: loss 5.918660
iteration 100 / 1500: loss 5.792131
iteration 200 / 1500: loss 6.314399
iteration 300 / 1500: loss 5.649129
iteration 400 / 1500: loss 5.075121
iteration 500 / 1500: loss 5.147106
iteration 600 / 1500: loss 5.577892
iteration 700 / 1500: loss 5.635790
iteration 800 / 1500: loss 5.586546
iteration 900 / 1500: loss 5.339323
iteration 1000 / 1500: loss 5.036398
iteration 1100 / 1500: loss 4.793873
iteration 1200 / 1500: loss 4.550206
iteration 1300 / 1500: loss 4.408830
iteration 1400 / 1500: loss 4.959526
Val_accuracy: 0.136, Current best_val: 0.4
count 341 / 600
iteration 0 / 20: loss 15.273554
count 342 / 600
iteration 0 / 20: loss 4.916176
iteration 0 / 1500: loss 5.865361
iteration 100 / 1500: loss 4.645946
iteration 200 / 1500: loss 4.200374
iteration 300 / 1500: loss 4.225685
iteration 400 / 1500: loss 3.339885
iteration 500 / 1500: loss 3.603790
iteration 600 / 1500: loss 3.589955
iteration 700 / 1500: loss 3.512970
iteration 800 / 1500: loss 4.023589
iteration 900 / 1500: loss 3.297427
iteration 1000 / 1500: loss 3.272511
iteration 1100 / 1500: loss 3.102879
iteration 1200 / 1500: loss 3.298063
iteration 1300 / 1500: loss 3.299118
iteration 1400 / 1500: loss 3.125435
Val_accuracy: 0.242, Current best_val: 0.4
count 343 / 600
iteration 0 / 20: loss 15.398766
count 344 / 600
iteration 0 / 20: loss 25.506672
count 345 / 600
iteration 0 / 20: loss 26.088989
count 346 / 600
iteration 0 / 20: loss 2580.638635
count 347 / 600
iteration 0 / 20: loss 8.307169
iteration 0 / 1500: loss 7.946956
iteration 100 / 1500: loss 5.042804
iteration 200 / 1500: loss 4.569372
iteration 300 / 1500: loss 4.148504
iteration 400 / 1500: loss 3.905891
iteration 500 / 1500: loss 3.839796
iteration 600 / 1500: loss 3.415342
iteration 700 / 1500: loss 3.550737
iteration 800 / 1500: loss 3.064257
iteration 900 / 1500: loss 3.163017
iteration 1000 / 1500: loss 3.107528
iteration 1100 / 1500: loss 3.086260
iteration 1200 / 1500: loss 2.818958
iteration 1300 / 1500: loss 2.628709
iteration 1400 / 1500: loss 2.767054
Val_accuracy: 0.4, Current best_val: 0.4
count 348 / 600
iteration 0 / 20: loss 889.120410
count 349 / 600
iteration 0 / 20: loss 881.166579
iteration 0 / 1500: loss 870.787273
iteration 100 / 1500: loss 3.944601
iteration 200 / 1500: loss 3.767770
iteration 300 / 1500: loss 4.638574
iteration 400 / 1500: loss 3.060168
iteration 500 / 1500: loss 3.866191
iteration 600 / 1500: loss 4.538714
iteration 700 / 1500: loss 2.628835
iteration 800 / 1500: loss 3.692139
iteration 900 / 1500: loss 3.220818
iteration 1000 / 1500: loss 4.809227
iteration 1100 / 1500: loss 3.744640
iteration 1200 / 1500: loss 3.096554
iteration 1300 / 1500: loss 3.158261
iteration 1400 / 1500: loss 4.024859
Val_accuracy: 0.197, Current best_val: 0.4
count 350 / 600
iteration 0 / 20: loss 6.013776
iteration 0 / 1500: loss 5.906262
iteration 100 / 1500: loss 5.599692
iteration 200 / 1500: loss 4.970167
iteration 300 / 1500: loss 4.524567
iteration 400 / 1500: loss 4.358684
iteration 500 / 1500: loss 4.480827
iteration 600 / 1500: loss 4.392469
iteration 700 / 1500: loss 4.511986
iteration 800 / 1500: loss 4.496869
iteration 900 / 1500: loss 4.201614
iteration 1000 / 1500: loss 4.010822
iteration 1100 / 1500: loss 4.337572
iteration 1200 / 1500: loss 4.109685
iteration 1300 / 1500: loss 4.028649
iteration 1400 / 1500: loss 3.998962
Val_accuracy: 0.144, Current best_val: 0.4
count 351 / 600
```

```
iteration 0 / 20: loss 5.313825
iteration 0 / 1500: loss 5.388460
iteration 100 / 1500: loss 3.632279
iteration 200 / 1500: loss 3.695868
iteration 300 / 1500: loss 3.338608
iteration 400 / 1500: loss 3.238968
iteration 500 / 1500: loss 2.940054
iteration 600 / 1500: loss 3.006032
iteration 700 / 1500: loss 2.689339
iteration 800 / 1500: loss 2.664295
iteration 900 / 1500: loss 2.976143
iteration 1000 / 1500: loss 2.579271
iteration 1100 / 1500: loss 2.544367
iteration 1200 / 1500: loss 2.795459
iteration 1300 / 1500: loss 2.771717
iteration 1400 / 1500: loss 2.762597
Val_accuracy: 0.267, Current best_val: 0.4
count 352 / 600
iteration 0 / 20: loss 627.610960
count 353 / 600
iteration 0 / 20: loss 6.101400
iteration 0 / 1500: loss 5.374906
iteration 100 / 1500: loss 5.118700
iteration 200 / 1500: loss 4.476392
iteration 300 / 1500: loss 4.320698
iteration 400 / 1500: loss 4.021970
iteration 500 / 1500: loss 3.960785
iteration 600 / 1500: loss 3.603584
iteration 700 / 1500: loss 3.597686
iteration 800 / 1500: loss 3.865659
iteration 900 / 1500: loss 3.685392
iteration 1000 / 1500: loss 3.958250
iteration 1100 / 1500: loss 3.668501
iteration 1200 / 1500: loss 3.295874
iteration 1300 / 1500: loss 3.636440
iteration 1400 / 1500: loss 3.444154
Val_accuracy: 0.166, Current best_val: 0.4
count 354 / 600
iteration 0 / 20: loss 8.110097
iteration 0 / 1500: loss 8.274066
iteration 100 / 1500: loss 6.887473
iteration 200 / 1500: loss 6.586405
iteration 300 / 1500: loss 6.240907
iteration 400 / 1500: loss 6.126438
iteration 500 / 1500: loss 6.027997
iteration 600 / 1500: loss 5.569760
iteration 700 / 1500: loss 5.713080
iteration 800 / 1500: loss 5.689350
iteration 900 / 1500: loss 5.262276
iteration 1000 / 1500: loss 4.955348
iteration 1100 / 1500: loss 5.289526
iteration 1200 / 1500: loss 5.258427
iteration 1300 / 1500: loss 4.859144
iteration 1400 / 1500: loss 4.936133
Val_accuracy: 0.185, Current best_val: 0.4
count 355 / 600
iteration 0 / 20: loss 7.909528
iteration 0 / 1500: loss 6.988091
iteration 100 / 1500: loss 6.384083
iteration 200 / 1500: loss 6.468903
iteration 300 / 1500: loss 5.786392
iteration 400 / 1500: loss 5.860037
iteration 500 / 1500: loss 6.043586
iteration 600 / 1500: loss 5.855412
iteration 700 / 1500: loss 5.666508
iteration 800 / 1500: loss 5.623556
iteration 900 / 1500: loss 6.048978
iteration 1000 / 1500: loss 5.612160
iteration 1100 / 1500: loss 5.495252
iteration 1200 / 1500: loss 5.360679
iteration 1300 / 1500: loss 5.609235
iteration 1400 / 1500: loss 5.459958
Val_accuracy: 0.185, Current best_val: 0.4
count 356 / 600
iteration 0 / 20: loss 6.554002
iteration 0 / 1500: loss 6.843862
iteration 100 / 1500: loss 4.464020
iteration 200 / 1500: loss 3.978634
iteration 300 / 1500: loss 3.999797
iteration 400 / 1500: loss 3.585848
iteration 500 / 1500: loss 3.414724
iteration 600 / 1500: loss 3.551208
iteration 700 / 1500: loss 3.675709
iteration 800 / 1500: loss 3.450358
iteration 900 / 1500: loss 3.318748
iteration 1000 / 1500: loss 3.276640
iteration 1100 / 1500: loss 3.330911
iteration 1200 / 1500: loss 3.300951
iteration 1300 / 1500: loss 3.089009
iteration 1400 / 1500: loss 3.278099
Val_accuracy: 0.279, Current best_val: 0.4
count 357 / 600
iteration 0 / 20: loss 8.351059
iteration 0 / 1500: loss 7.693858
iteration 100 / 1500: loss 5.725320
iteration 200 / 1500: loss 5.146499
iteration 300 / 1500: loss 4.848799
iteration 400 / 1500: loss 4.624338
iteration 500 / 1500: loss 4.637946
iteration 600 / 1500: loss 4.813160
iteration 700 / 1500: loss 4.385764
iteration 800 / 1500: loss 4.260833
iteration 900 / 1500: loss 4.322044
iteration 1000 / 1500: loss 4.249306
iteration 1100 / 1500: loss 4.111449
iteration 1200 / 1500: loss 4.071612
iteration 1300 / 1500: loss 4.312643
iteration 1400 / 1500: loss 4.239216
Val_accuracy: 0.305, Current best_val: 0.4
count 358 / 600
iteration 0 / 20: loss 5.665431
iteration 0 / 1500: loss 5.697446
iteration 100 / 1500: loss 3.693213
iteration 200 / 1500: loss 3.078709
iteration 300 / 1500: loss 3.319558
iteration 400 / 1500: loss 2.904681
iteration 500 / 1500: loss 2.705374
iteration 600 / 1500: loss 2.622860
iteration 700 / 1500: loss 2.516906
iteration 800 / 1500: loss 2.562664
iteration 900 / 1500: loss 2.459098
iteration 1000 / 1500: loss 2.503782
iteration 1100 / 1500: loss 2.805136
iteration 1200 / 1500: loss 2.305538
iteration 1300 / 1500: loss 2.595802
iteration 1400 / 1500: loss 2.612777
Val_accuracy: 0.283, Current best_val: 0.4
count 359 / 600
iteration 0 / 20: loss 2329.278101
count 360 / 600
iteration 0 / 20: loss 52.249679
count 361 / 600
iteration 0 / 20: loss 6.192973
iteration 0 / 1500: loss 4.746963
iteration 100 / 1500: loss 3.064214
iteration 200 / 1500: loss 2.869004
iteration 300 / 1500: loss 2.637039
iteration 400 / 1500: loss 2.466817
iteration 500 / 1500: loss 2.616921
iteration 600 / 1500: loss 2.365193
iteration 700 / 1500: loss 2.305437
iteration 800 / 1500: loss 2.143517
iteration 900 / 1500: loss 2.189376
iteration 1000 / 1500: loss 2.189636
iteration 1100 / 1500: loss 2.345184
iteration 1200 / 1500: loss 2.110809
iteration 1300 / 1500: loss 1.960334
iteration 1400 / 1500: loss 2.135116
Val_accuracy: 0.296, Current best_val: 0.4
count 362 / 600
iteration 0 / 20: loss 6.220504
iteration 0 / 1500: loss 5.975104
iteration 100 / 1500: loss 3.567176
iteration 200 / 1500: loss 2.887268
iteration 300 / 1500: loss 3.007625
iteration 400 / 1500: loss 2.899148
iteration 500 / 1500: loss 2.700574
iteration 600 / 1500: loss 2.558385
iteration 700 / 1500: loss 2.590285
iteration 800 / 1500: loss 2.666615
iteration 900 / 1500: loss 2.552163
iteration 1000 / 1500: loss 2.517927
iteration 1100 / 1500: loss 2.432145
iteration 1200 / 1500: loss 2.454397
iteration 1300 / 1500: loss 2.372595
iteration 1400 / 1500: loss 2.421483
Val_accuracy: 0.357, Current best_val: 0.4
count 363 / 600
iteration 0 / 20: loss 174.835428
count 364 / 600
iteration 0 / 20: loss 1264.128681
count 365 / 600
iteration 0 / 20: loss 118.866758
count 366 / 600
iteration 0 / 20: loss 6.608199
iteration 0 / 1500: loss 6.130139
iteration 100 / 1500: loss 6.238545
iteration 200 / 1500: loss 6.165194
iteration 300 / 1500: loss 5.482071
iteration 400 / 1500: loss 5.614746
iteration 500 / 1500: loss 5.536636
iteration 600 / 1500: loss 5.374554
iteration 700 / 1500: loss 5.466653
iteration 800 / 1500: loss 5.238791
iteration 900 / 1500: loss 5.044000
iteration 1000 / 1500: loss 5.060932
iteration 1100 / 1500: loss 4.883114
iteration 1200 / 1500: loss 5.202515
iteration 1300 / 1500: loss 4.799701
iteration 1400 / 1500: loss 4.815942
Val_accuracy: 0.102, Current best_val: 0.4
count 367 / 600
```

```
iteration 0 / 20: loss 1709.568803
count 368 / 600
iteration 0 / 20: loss 6.250051
iteration 0 / 1500: loss 6.169155
iteration 100 / 1500: loss 6.260035
iteration 200 / 1500: loss 5.465156
iteration 300 / 1500: loss 5.686517
iteration 400 / 1500: loss 5.995625
iteration 500 / 1500: loss 5.711012
iteration 600 / 1500: loss 5.583657
iteration 700 / 1500: loss 5.287074
iteration 800 / 1500: loss 5.224881
iteration 900 / 1500: loss 4.818665
iteration 1000 / 1500: loss 5.043227
iteration 1100 / 1500: loss 5.545395
iteration 1200 / 1500: loss 5.077303
iteration 1300 / 1500: loss 5.236558
iteration 1400 / 1500: loss 4.886884
Val_accuracy: 0.111, Current best_val: 0.4
count 369 / 600
iteration 0 / 20: loss 135.140616
count 370 / 600
iteration 0 / 20: loss 440.792928
count 371 / 600
iteration 0 / 20: loss 192.361154
count 372 / 600
iteration 0 / 20: loss 56.203661
count 373 / 600
iteration 0 / 20: loss 5.501249
iteration 0 / 1500: loss 5.671215
iteration 100 / 1500: loss 5.930677
iteration 200 / 1500: loss 5.829791
iteration 300 / 1500: loss 5.839417
iteration 400 / 1500: loss 5.757929
iteration 500 / 1500: loss 5.836222
iteration 600 / 1500: loss 5.844863
iteration 700 / 1500: loss 5.582945
iteration 800 / 1500: loss 5.105649
iteration 900 / 1500: loss 5.256870
iteration 1000 / 1500: loss 5.341394
iteration 1100 / 1500: loss 5.871154
iteration 1200 / 1500: loss 5.712555
iteration 1300 / 1500: loss 5.850101
iteration 1400 / 1500: loss 5.522079
Val_accuracy: 0.103, Current best_val: 0.4
count 374 / 600
iteration 0 / 20: loss 466.154728
count 375 / 600
```

```
iteration 0 / 20: loss 7.309843
iteration 0 / 1500: loss 7.095970
iteration 100 / 1500: loss 6.297520
iteration 200 / 1500: loss 5.979376
iteration 300 / 1500: loss 5.981210
iteration 400 / 1500: loss 5.847205
iteration 500 / 1500: loss 5.581593
iteration 600 / 1500: loss 4.888945
iteration 700 / 1500: loss 5.175601
iteration 800 / 1500: loss 5.011133
iteration 900 / 1500: loss 4.945051
iteration 1000 / 1500: loss 4.803899
iteration 1100 / 1500: loss 5.111640
iteration 1200 / 1500: loss 4.990146
iteration 1300 / 1500: loss 4.934577
iteration 1400 / 1500: loss 4.726032
Val_accuracy: 0.2, Current best_val: 0.4
count 376 / 600
iteration 0 / 20: loss 570.077684
count 377 / 600
iteration 0 / 20: loss 2482.639272
count 378 / 600
iteration 0 / 20: loss 40.187212
count 379 / 600
iteration 0 / 20: loss 86.659658
count 380 / 600
iteration 0 / 20: loss 14.846173
count 381 / 600
iteration 0 / 20: loss 7.420332
iteration 0 / 1500: loss 9.009553
iteration 100 / 1500: loss 6.732299
iteration 200 / 1500: loss 6.112063
iteration 300 / 1500: loss 5.856727
iteration 400 / 1500: loss 5.662358
iteration 500 / 1500: loss 5.627109
iteration 600 / 1500: loss 5.683212
iteration 700 / 1500: loss 5.441383
iteration 800 / 1500: loss 5.342984
iteration 900 / 1500: loss 4.959052
iteration 1000 / 1500: loss 5.154312
iteration 1100 / 1500: loss 5.085867
iteration 1200 / 1500: loss 4.829887
iteration 1300 / 1500: loss 4.682871
iteration 1400 / 1500: loss 4.749547
Val_accuracy: 0.313, Current best_val: 0.4
count 382 / 600
iteration 0 / 20: loss 6.544824
iteration 0 / 1500: loss 6.423982
iteration 100 / 1500: loss 4.911841
iteration 200 / 1500: loss 4.127315
iteration 300 / 1500: loss 4.290323
iteration 400 / 1500: loss 3.798825
iteration 500 / 1500: loss 3.836759
iteration 600 / 1500: loss 4.019240
iteration 700 / 1500: loss 3.614413
iteration 800 / 1500: loss 3.497155
iteration 900 / 1500: loss 3.510660
iteration 1000 / 1500: loss 3.341497
iteration 1100 / 1500: loss 3.247921
iteration 1200 / 1500: loss 3.425899
iteration 1300 / 1500: loss 3.479809
iteration 1400 / 1500: loss 3.360344
Val_accuracy: 0.237, Current best_val: 0.4
count 383 / 600
iteration 0 / 20: loss 6.099541
iteration 0 / 1500: loss 5.668068
iteration 100 / 1500: loss 4.279983
iteration 200 / 1500: loss 3.911883
iteration 300 / 1500: loss 3.480677
iteration 400 / 1500: loss 3.396688
iteration 500 / 1500: loss 3.468027
iteration 600 / 1500: loss 3.017181
iteration 700 / 1500: loss 2.940067
iteration 800 / 1500: loss 2.861306
iteration 900 / 1500: loss 2.851794
iteration 1000 / 1500: loss 2.931707
iteration 1100 / 1500: loss 2.686064
iteration 1200 / 1500: loss 2.574503
iteration 1300 / 1500: loss 2.756696
iteration 1400 / 1500: loss 2.536495
Val_accuracy: 0.246, Current best_val: 0.4
count 384 / 600
iteration 0 / 20: loss 5.159946
iteration 0 / 1500: loss 5.074134
iteration 100 / 1500: loss 5.457498
iteration 200 / 1500: loss 4.982547
iteration 300 / 1500: loss 5.350209
iteration 400 / 1500: loss 5.021217
iteration 500 / 1500: loss 4.514616
iteration 600 / 1500: loss 4.587836
iteration 700 / 1500: loss 4.692722
iteration 800 / 1500: loss 4.396935
iteration 900 / 1500: loss 4.569401
iteration 1000 / 1500: loss 4.802351
iteration 1100 / 1500: loss 4.437677
iteration 1200 / 1500: loss 4.599312
iteration 1300 / 1500: loss 4.147921
iteration 1400 / 1500: loss 3.875467
Val_accuracy: 0.132, Current best_val: 0.4
count 385 / 600
iteration 0 / 20: loss 5.382376
iteration 0 / 1500: loss 5.991423
iteration 100 / 1500: loss 3.523404
iteration 200 / 1500: loss 3.310408
iteration 300 / 1500: loss 3.152956
iteration 400 / 1500: loss 2.755725
iteration 500 / 1500: loss 3.098212
iteration 600 / 1500: loss 2.878897
iteration 700 / 1500: loss 2.767225
iteration 800 / 1500: loss 2.566852
iteration 900 / 1500: loss 2.749643
iteration 1000 / 1500: loss 2.547639
iteration 1100 / 1500: loss 2.343167
iteration 1200 / 1500: loss 2.636437
iteration 1300 / 1500: loss 2.449278
iteration 1400 / 1500: loss 2.426950
Val_accuracy: 0.294, Current best_val: 0.4
count 386 / 600
iteration 0 / 20: loss 15.890906
count 387 / 600
iteration 0 / 20: loss 5.991707
iteration 0 / 1500: loss 5.016164
iteration 100 / 1500: loss 4.212043
iteration 200 / 1500: loss 3.634884
iteration 300 / 1500: loss 3.867062
iteration 400 / 1500: loss 3.615965
iteration 500 / 1500: loss 3.413532
iteration 600 / 1500: loss 3.460773
iteration 700 / 1500: loss 3.647232
iteration 800 / 1500: loss 3.062685
iteration 900 / 1500: loss 3.055847
iteration 1000 / 1500: loss 2.975323
iteration 1100 / 1500: loss 3.176041
iteration 1200 / 1500: loss 3.048160
iteration 1300 / 1500: loss 3.148841
iteration 1400 / 1500: loss 3.237741
Val_accuracy: 0.223, Current best_val: 0.4
count 388 / 600
iteration 0 / 20: loss 59.461643
count 389 / 600
iteration 0 / 20: loss 17.615240
count 390 / 600
iteration 0 / 20: loss 6.343951
iteration 0 / 1500: loss 6.290619
iteration 100 / 1500: loss 3.853907
iteration 200 / 1500: loss 3.630174
iteration 300 / 1500: loss 3.456249
iteration 400 / 1500: loss 3.187485
iteration 500 / 1500: loss 3.206750
iteration 600 / 1500: loss 3.092461
iteration 700 / 1500: loss 3.037496
iteration 800 / 1500: loss 3.170449
iteration 900 / 1500: loss 2.933707
iteration 1000 / 1500: loss 3.131782
iteration 1100 / 1500: loss 2.998420
iteration 1200 / 1500: loss 3.101636
iteration 1300 / 1500: loss 2.841762
iteration 1400 / 1500: loss 2.905909
Val_accuracy: 0.311, Current best_val: 0.4
count 391 / 600
iteration 0 / 20: loss 5.610270
iteration 0 / 1500: loss 6.015637
iteration 100 / 1500: loss 5.157234
iteration 200 / 1500: loss 4.834579
iteration 300 / 1500: loss 4.591145
iteration 400 / 1500: loss 4.424468
iteration 500 / 1500: loss 3.974209
iteration 600 / 1500: loss 3.687800
iteration 700 / 1500: loss 4.019460
iteration 800 / 1500: loss 3.412507
iteration 900 / 1500: loss 3.875111
iteration 1000 / 1500: loss 3.653229
iteration 1100 / 1500: loss 3.731861
iteration 1200 / 1500: loss 3.628856
iteration 1300 / 1500: loss 3.455841
iteration 1400 / 1500: loss 3.409002
Val_accuracy: 0.169, Current best_val: 0.4
count 392 / 600
iteration 0 / 20: loss 19.345371
count 393 / 600
iteration 0 / 20: loss 262.985707
count 394 / 600
iteration 0 / 20: loss 14.869890
iteration 0 / 1500: loss 15.399755
iteration 100 / 1500: loss 10.367899
iteration 200 / 1500: loss 9.392668
iteration 300 / 1500: loss 8.281538
iteration 400 / 1500: loss 7.446001
iteration 500 / 1500: loss 6.870917
iteration 600 / 1500: loss 6.427490
iteration 700 / 1500: loss 5.815799
iteration 800 / 1500: loss 5.326585
iteration 900 / 1500: loss 4.919025
iteration 1000 / 1500: loss 4.428204
iteration 1100 / 1500: loss 4.145580
iteration 1200 / 1500: loss 4.084825
iteration 1300 / 1500: loss 3.826925
iteration 1400 / 1500: loss 3.621837
Val_accuracy: 0.384, Current best_val: 0.4
count 395 / 600
iteration 0 / 20: loss 6.571674
iteration 0 / 1500: loss 6.616034
iteration 100 / 1500: loss 4.026461
iteration 200 / 1500: loss 3.436385
iteration 300 / 1500: loss 3.090266
iteration 400 / 1500: loss 3.183329
iteration 500 / 1500: loss 3.059039
iteration 600 / 1500: loss 2.788410
iteration 700 / 1500: loss 3.176500
iteration 800 / 1500: loss 3.037230
iteration 900 / 1500: loss 2.927778
iteration 1000 / 1500: loss 2.686976
iteration 1100 / 1500: loss 2.740132
iteration 1200 / 1500: loss 2.819802
iteration 1300 / 1500: loss 2.726022
iteration 1400 / 1500: loss 2.669093
Val_accuracy: 0.326, Current best_val: 0.4
count 396 / 600
iteration 0 / 20: loss 5.734406
iteration 0 / 1500: loss 5.825040
iteration 100 / 1500: loss 3.074935
iteration 200 / 1500: loss 2.991730
iteration 300 / 1500: loss 2.510516
iteration 400 / 1500: loss 2.734398
iteration 500 / 1500: loss 2.514252
iteration 600 / 1500: loss 2.484326
iteration 700 / 1500: loss 2.412456
iteration 800 / 1500: loss 2.375982
iteration 900 / 1500: loss 2.434370
iteration 1000 / 1500: loss 2.341351
iteration 1100 / 1500: loss 2.374102
iteration 1200 / 1500: loss 2.238226
iteration 1300 / 1500: loss 2.605894
iteration 1400 / 1500: loss 2.444171
Val_accuracy: 0.327, Current best_val: 0.4
count 397 / 600
iteration 0 / 20: loss 19.489936
count 398 / 600
iteration 0 / 20: loss 53.816930
count 399 / 600
```

```
iteration 0 / 20: loss 5.800523
iteration 0 / 1500: loss 5.286283
iteration 100 / 1500: loss 2.078475
iteration 200 / 1500: loss 1.979606
iteration 300 / 1500: loss 2.172736
iteration 400 / 1500: loss 1.928847
iteration 500 / 1500: loss 1.942068
iteration 600 / 1500: loss 1.965442
iteration 700 / 1500: loss 1.983806
iteration 800 / 1500: loss 1.853099
iteration 900 / 1500: loss 2.058865
iteration 1000 / 1500: loss 1.913633
iteration 1100 / 1500: loss 1.997126
iteration 1200 / 1500: loss 1.785286
iteration 1300 / 1500: loss 1.692121
iteration 1400 / 1500: loss 1.870260
Val_accuracy: 0.367, Current best_val: 0.4
count 400 / 600
iteration 0 / 20: loss 36.737050
count 401 / 600
iteration 0 / 20: loss 6.610047
iteration 0 / 1500: loss 6.213887
iteration 100 / 1500: loss 2.802744
iteration 200 / 1500: loss 2.575895
iteration 300 / 1500: loss 2.437510
iteration 400 / 1500: loss 2.326327
iteration 500 / 1500: loss 2.425907
iteration 600 / 1500: loss 2.249786
iteration 700 / 1500: loss 2.388779
iteration 800 / 1500: loss 2.176781
iteration 900 / 1500: loss 2.314307
iteration 1000 / 1500: loss 2.199940
iteration 1100 / 1500: loss 2.216533
iteration 1200 / 1500: loss 2.260934
iteration 1300 / 1500: loss 2.066303
iteration 1400 / 1500: loss 2.129807
Val_accuracy: 0.364, Current best_val: 0.4
count 402 / 600
iteration 0 / 20: loss 6.130991
iteration 0 / 1500: loss 5.783366
iteration 100 / 1500: loss 4.974714
iteration 200 / 1500: loss 4.778556
iteration 300 / 1500: loss 4.288745
iteration 400 / 1500: loss 4.261478
iteration 500 / 1500: loss 3.924575
iteration 600 / 1500: loss 4.090663
iteration 700 / 1500: loss 3.674378
iteration 800 / 1500: loss 3.581969
iteration 900 / 1500: loss 3.369065
iteration 1000 / 1500: loss 3.549904
iteration 1100 / 1500: loss 3.304084
iteration 1200 / 1500: loss 3.478502
iteration 1300 / 1500: loss 3.452682
iteration 1400 / 1500: loss 3.342176
Val_accuracy: 0.213, Current best_val: 0.4
count 403 / 600
iteration 0 / 20: loss 199.442726
count 404 / 600
iteration 0 / 20: loss 87.641936
count 405 / 600
iteration 0 / 20: loss 44.549071
count 406 / 600
iteration 0 / 20: loss 5.527845
iteration 0 / 1500: loss 5.549925
iteration 100 / 1500: loss 3.611852
iteration 200 / 1500: loss 3.323847
iteration 300 / 1500: loss 2.975833
iteration 400 / 1500: loss 2.906200
iteration 500 / 1500: loss 2.898666
iteration 600 / 1500: loss 2.697067
iteration 700 / 1500: loss 2.650207
iteration 800 / 1500: loss 2.599300
iteration 900 / 1500: loss 2.603984
iteration 1000 / 1500: loss 2.862978
iteration 1100 / 1500: loss 2.410730
iteration 1200 / 1500: loss 2.581901
iteration 1300 / 1500: loss 2.658269
iteration 1400 / 1500: loss 2.537091
Val_accuracy: 0.271, Current best_val: 0.4
count 407 / 600
iteration 0 / 20: loss 114.073083
count 408 / 600
iteration 0 / 20: loss 85.404270
count 409 / 600
iteration 0 / 20: loss 5.453748
iteration 0 / 1500: loss 5.713462
iteration 100 / 1500: loss 2.931689
iteration 200 / 1500: loss 2.586006
iteration 300 / 1500: loss 2.249617
iteration 400 / 1500: loss 2.623262
iteration 500 / 1500: loss 2.151592
iteration 600 / 1500: loss 2.276945
iteration 700 / 1500: loss 2.121497
iteration 800 / 1500: loss 2.001942
iteration 900 / 1500: loss 2.132727
iteration 1000 / 1500: loss 1.893870
iteration 1100 / 1500: loss 1.970334
iteration 1200 / 1500: loss 2.080447
iteration 1300 / 1500: loss 2.111410
iteration 1400 / 1500: loss 2.098274
Val_accuracy: 0.338, Current best_val: 0.4
count 410 / 600
iteration 0 / 20: loss 875.058799
count 411 / 600
iteration 0 / 20: loss 60.069677
count 412 / 600
iteration 0 / 20: loss 19.616769
count 413 / 600
iteration 0 / 20: loss 5.872263
iteration 0 / 1500: loss 6.090531
iteration 100 / 1500: loss 2.952600
iteration 200 / 1500: loss 2.619291
iteration 300 / 1500: loss 2.390407
iteration 400 / 1500: loss 2.168484
iteration 500 / 1500: loss 2.193821
iteration 600 / 1500: loss 2.042629
iteration 700 / 1500: loss 2.055380
iteration 800 / 1500: loss 2.045155
iteration 900 / 1500: loss 1.974871
iteration 1000 / 1500: loss 1.999931
iteration 1100 / 1500: loss 2.020138
iteration 1200 / 1500: loss 2.026191
iteration 1300 / 1500: loss 1.909971
iteration 1400 / 1500: loss 2.061391
Val_accuracy: 0.323, Current best_val: 0.4
count 414 / 600
iteration 0 / 20: loss 118.208111
count 415 / 600
iteration 0 / 20: loss 1768.609935
count 416 / 600
iteration 0 / 20: loss 1353.514668
count 417 / 600
iteration 0 / 20: loss 10.233288
iteration 0 / 1500: loss 11.306575
iteration 100 / 1500: loss 9.197136
iteration 200 / 1500: loss 8.854189
iteration 300 / 1500: loss 8.296824
iteration 400 / 1500: loss 8.303422
iteration 500 / 1500: loss 7.720890
iteration 600 / 1500: loss 7.403629
iteration 700 / 1500: loss 7.548685
iteration 800 / 1500: loss 7.442549
iteration 900 / 1500: loss 7.197165
iteration 1000 / 1500: loss 7.288629
iteration 1100 / 1500: loss 7.252689
iteration 1200 / 1500: loss 7.161897
iteration 1300 / 1500: loss 6.844598
iteration 1400 / 1500: loss 7.087804
Val_accuracy: 0.262, Current best_val: 0.4
count 418 / 600
iteration 0 / 20: loss 5.820808
iteration 0 / 1500: loss 5.426511
iteration 100 / 1500: loss 4.577963
iteration 200 / 1500: loss 5.145070
iteration 300 / 1500: loss 4.216844
iteration 400 / 1500: loss 4.241065
iteration 500 / 1500: loss 3.919426
iteration 600 / 1500: loss 4.013073
iteration 700 / 1500: loss 3.915049
iteration 800 / 1500: loss 4.162162
iteration 900 / 1500: loss 3.782166
iteration 1000 / 1500: loss 4.045020
iteration 1100 / 1500: loss 3.868258
iteration 1200 / 1500: loss 3.613660
iteration 1300 / 1500: loss 3.767070
iteration 1400 / 1500: loss 3.969214
Val_accuracy: 0.18, Current best_val: 0.4
count 419 / 600
iteration 0 / 20: loss 5.753761
iteration 0 / 1500: loss 5.866050
iteration 100 / 1500: loss 5.335078
iteration 200 / 1500: loss 4.912890
iteration 300 / 1500: loss 4.751115
iteration 400 / 1500: loss 5.090743
iteration 500 / 1500: loss 4.707550
iteration 600 / 1500: loss 5.108464
iteration 700 / 1500: loss 4.761769
iteration 800 / 1500: loss 4.774758
iteration 900 / 1500: loss 4.453045
iteration 1000 / 1500: loss 4.525814
iteration 1100 / 1500: loss 4.336202
iteration 1200 / 1500: loss 4.410015
iteration 1300 / 1500: loss 4.255927
iteration 1400 / 1500: loss 4.364683
Val_accuracy: 0.16, Current best_val: 0.4
count 420 / 600
iteration 0 / 20: loss 6.008072
iteration 0 / 1500: loss 5.894666
iteration 100 / 1500: loss 2.981986
iteration 200 / 1500: loss 2.846881
iteration 300 / 1500: loss 2.491550
iteration 400 / 1500: loss 2.538646
iteration 500 / 1500: loss 2.631499
iteration 600 / 1500: loss 2.203097
iteration 700 / 1500: loss 2.366174
iteration 800 / 1500: loss 2.396394
iteration 900 / 1500: loss 2.331706
iteration 1000 / 1500: loss 2.290675
iteration 1100 / 1500: loss 2.270927
iteration 1200 / 1500: loss 2.123560
iteration 1300 / 1500: loss 2.048071
iteration 1400 / 1500: loss 2.206431
Val_accuracy: 0.34, Current best_val: 0.4
count 421 / 600
iteration 0 / 20: loss 99.713903
count 422 / 600
iteration 0 / 20: loss 1251.412732
iteration 0 / 1500: loss 1274.197157
iteration 100 / 1500: loss 5.078900
iteration 200 / 1500: loss 6.144910
iteration 300 / 1500: loss 5.885442
iteration 400 / 1500: loss 6.177649
iteration 500 / 1500: loss 5.878718
iteration 600 / 1500: loss 5.950753
iteration 700 / 1500: loss 8.395829
iteration 800 / 1500: loss 5.580521
iteration 900 / 1500: loss 4.798221
iteration 1000 / 1500: loss 5.002416
iteration 1100 / 1500: loss 6.389832
iteration 1200 / 1500: loss 5.298060
iteration 1300 / 1500: loss 5.843222
iteration 1400 / 1500: loss 5.740911
Val_accuracy: 0.17, Current best_val: 0.4
count 423 / 600
iteration 0 / 20: loss 960.244992
count 424 / 600
iteration 0 / 20: loss 17.726526
count 425 / 600
iteration 0 / 20: loss 5.828063
iteration 0 / 1500: loss 6.708185
iteration 100 / 1500: loss 5.345012
iteration 200 / 1500: loss 5.872123
iteration 300 / 1500: loss 6.332158
iteration 400 / 1500: loss 5.900429
iteration 500 / 1500: loss 6.192589
iteration 600 / 1500: loss 5.689755
iteration 700 / 1500: loss 5.582757
iteration 800 / 1500: loss 6.084476
iteration 900 / 1500: loss 6.081333
iteration 1000 / 1500: loss 5.979088
iteration 1100 / 1500: loss 5.471049
iteration 1200 / 1500: loss 5.898323
iteration 1300 / 1500: loss 5.483946
iteration 1400 / 1500: loss 6.756471
Val_accuracy: 0.1, Current best_val: 0.4
count 426 / 600
iteration 0 / 20: loss 5.190324
iteration 0 / 1500: loss 5.427257
iteration 100 / 1500: loss 4.912864
iteration 200 / 1500: loss 4.236135
iteration 300 / 1500: loss 4.364079
iteration 400 / 1500: loss 3.828711
iteration 500 / 1500: loss 3.569908
iteration 600 / 1500: loss 3.721163
iteration 700 / 1500: loss 3.488085
iteration 800 / 1500: loss 3.718487
iteration 900 / 1500: loss 3.692680
iteration 1000 / 1500: loss 3.275208
iteration 1100 / 1500: loss 3.244061
iteration 1200 / 1500: loss 3.445686
iteration 1300 / 1500: loss 3.318418
iteration 1400 / 1500: loss 3.075431
Val_accuracy: 0.184, Current best_val: 0.4
count 427 / 600
iteration 0 / 20: loss 47.808331
count 428 / 600
iteration 0 / 20: loss 6.467656
iteration 0 / 1500: loss 6.547281
iteration 100 / 1500: loss 5.012449
iteration 200 / 1500: loss 4.505877
iteration 300 / 1500: loss 4.233871
iteration 400 / 1500: loss 3.769331
iteration 500 / 1500: loss 3.974828
iteration 600 / 1500: loss 3.689622
iteration 700 / 1500: loss 3.563339
iteration 800 / 1500: loss 3.664139
iteration 900 / 1500: loss 3.320541
iteration 1000 / 1500: loss 3.415119
iteration 1100 / 1500: loss 3.266820
iteration 1200 / 1500: loss 3.184490
iteration 1300 / 1500: loss 3.205435
iteration 1400 / 1500: loss 3.409127
Val_accuracy: 0.219, Current best_val: 0.4
count 429 / 600
iteration 0 / 20: loss 52.342508
count 430 / 600
iteration 0 / 20: loss 5.353314
iteration 0 / 1500: loss 4.680041
iteration 100 / 1500: loss 5.402775
iteration 200 / 1500: loss 5.014240
iteration 300 / 1500: loss 5.146074
iteration 400 / 1500: loss 4.645100
iteration 500 / 1500: loss 4.573011
iteration 600 / 1500: loss 4.634259
iteration 700 / 1500: loss 4.923298
iteration 800 / 1500: loss 4.450983
iteration 900 / 1500: loss 4.531984
iteration 1000 / 1500: loss 4.483004
iteration 1100 / 1500: loss 4.383169
iteration 1200 / 1500: loss 4.504265
iteration 1300 / 1500: loss 4.812641
iteration 1400 / 1500: loss 4.354179
Val_accuracy: 0.168, Current best_val: 0.4
count 431 / 600
iteration 0 / 20: loss 1375.392902
iteration 0 / 1500: loss 1383.907837
iteration 100 / 1500: loss 9.711829
iteration 200 / 1500: loss 10.018186
iteration 300 / 1500: loss 9.562978
iteration 400 / 1500: loss 10.028624
iteration 500 / 1500: loss 8.011386
iteration 600 / 1500: loss 9.994350
iteration 700 / 1500: loss 8.085716
iteration 800 / 1500: loss 9.020758
iteration 900 / 1500: loss 13.051356
iteration 1000 / 1500: loss 9.247996
iteration 1100 / 1500: loss 9.800153
iteration 1200 / 1500: loss 9.347613
iteration 1300 / 1500: loss 8.207859
iteration 1400 / 1500: loss 9.310271
Val_accuracy: 0.093, Current best_val: 0.4
count 432 / 600
iteration 0 / 20: loss 5.356198
iteration 0 / 1500: loss 5.926570
iteration 100 / 1500: loss 5.947536
iteration 200 / 1500: loss 6.351495
iteration 300 / 1500: loss 6.150812
iteration 400 / 1500: loss 5.945401
iteration 500 / 1500: loss 5.961813
iteration 600 / 1500: loss 5.694592
iteration 700 / 1500: loss 5.291922
iteration 800 / 1500: loss 6.453349
iteration 900 / 1500: loss 6.185684
iteration 1000 / 1500: loss 5.575295
iteration 1100 / 1500: loss 6.023087
iteration 1200 / 1500: loss 5.965805
iteration 1300 / 1500: loss 5.410819
iteration 1400 / 1500: loss 5.326362
Val_accuracy: 0.083, Current best_val: 0.4
count 433 / 600
iteration 0 / 20: loss 5.876900
iteration 0 / 1500: loss 5.854451
iteration 100 / 1500: loss 5.204033
iteration 200 / 1500: loss 4.192662
iteration 300 / 1500: loss 4.258356
iteration 400 / 1500: loss 3.800997
iteration 500 / 1500: loss 3.525878
iteration 600 / 1500: loss 3.838052
iteration 700 / 1500: loss 3.702399
iteration 800 / 1500: loss 3.408981
iteration 900 / 1500: loss 3.522663
iteration 1000 / 1500: loss 3.214984
iteration 1100 / 1500: loss 3.431767
iteration 1200 / 1500: loss 3.196024
iteration 1300 / 1500: loss 3.110639
iteration 1400 / 1500: loss 3.005363
Val_accuracy: 0.236, Current best_val: 0.4
count 434 / 600
iteration 0 / 20: loss 6.512297
iteration 0 / 1500: loss 7.062816
iteration 100 / 1500: loss 5.308932
iteration 200 / 1500: loss 5.330729
iteration 300 / 1500: loss 5.056155
iteration 400 / 1500: loss 4.376358
iteration 500 / 1500: loss 4.844121
iteration 600 / 1500: loss 4.443868
iteration 700 / 1500: loss 4.182664
iteration 800 / 1500: loss 4.363349
iteration 900 / 1500: loss 4.363550
iteration 1000 / 1500: loss 3.963923
iteration 1100 / 1500: loss 4.041250
iteration 1200 / 1500: loss 4.031775
iteration 1300 / 1500: loss 4.242935
iteration 1400 / 1500: loss 4.010781
Val_accuracy: 0.224, Current best_val: 0.4
count 435 / 600
iteration 0 / 20: loss 684.427804
iteration 0 / 1500: loss 690.066879
iteration 100 / 1500: loss 2.130647
iteration 200 / 1500: loss 2.109386
iteration 300 / 1500: loss 2.127318
iteration 400 / 1500: loss 2.086819
iteration 500 / 1500: loss 2.163005
iteration 600 / 1500: loss 2.071145
iteration 700 / 1500: loss 2.168683
iteration 800 / 1500: loss 2.181147
iteration 900 / 1500: loss 2.092838
iteration 1000 / 1500: loss 2.142193
iteration 1100 / 1500: loss 2.036291
iteration 1200 / 1500: loss 2.156864
iteration 1300 / 1500: loss 2.204937
iteration 1400 / 1500: loss 2.064534
Val_accuracy: 0.291, Current best_val: 0.4
count 436 / 600
iteration 0 / 20: loss 262.435744
count 437 / 600
iteration 0 / 20: loss 5.798640
iteration 0 / 1500: loss 6.430055
iteration 100 / 1500: loss 4.511784
iteration 200 / 1500: loss 4.311129
iteration 300 / 1500: loss 3.850260
iteration 400 / 1500: loss 3.821366
iteration 500 / 1500: loss 3.657017
iteration 600 / 1500: loss 3.642947
iteration 700 / 1500: loss 3.420516
iteration 800 / 1500: loss 3.453867
iteration 900 / 1500: loss 3.212080
iteration 1000 / 1500: loss 3.394375
iteration 1100 / 1500: loss 2.971673
iteration 1200 / 1500: loss 3.250312
iteration 1300 / 1500: loss 3.243384
iteration 1400 / 1500: loss 3.141652
Val_accuracy: 0.255, Current best_val: 0.4
count 438 / 600
iteration 0 / 20: loss 270.836782
count 439 / 600
iteration 0 / 20: loss 5.864793
iteration 0 / 1500: loss 5.360279
iteration 100 / 1500: loss 5.008662
iteration 200 / 1500: loss 4.629881
iteration 300 / 1500: loss 4.266689
iteration 400 / 1500: loss 4.064490
iteration 500 / 1500: loss 3.864184
iteration 600 / 1500: loss 3.890302
iteration 700 / 1500: loss 3.650836
iteration 800 / 1500: loss 4.007041
iteration 900 / 1500: loss 3.545121
iteration 1000 / 1500: loss 3.317453
iteration 1100 / 1500: loss 3.558867
iteration 1200 / 1500: loss 3.673605
iteration 1300 / 1500: loss 3.454547
iteration 1400 / 1500: loss 3.319629
```

```
Val_accuracy: 0.189, Current best_val: 0.4
count 440 / 600
iteration 0 / 20: loss 1327.459749
count 441 / 600
iteration 0 / 20: loss 5.485276
iteration 0 / 1500: loss 5.585008
iteration 100 / 1500: loss 2.679876
iteration 200 / 1500: loss 2.426876
iteration 300 / 1500: loss 2.417463
iteration 400 / 1500: loss 2.109819
iteration 500 / 1500: loss 2.234560
iteration 600 / 1500: loss 2.268305
iteration 700 / 1500: loss 1.915461
iteration 800 / 1500: loss 2.130319
iteration 900 / 1500: loss 2.159318
iteration 1000 / 1500: loss 1.949396
iteration 1100 / 1500: loss 2.045126
iteration 1200 / 1500: loss 1.863382
iteration 1300 / 1500: loss 1.869756
iteration 1400 / 1500: loss 1.995195
Val_accuracy: 0.372, Current best_val: 0.4
count 442 / 600
iteration 0 / 20: loss 5.340487
iteration 0 / 1500: loss 5.429260
iteration 100 / 1500: loss 3.812139
iteration 200 / 1500: loss 3.404083
iteration 300 / 1500: loss 3.060009
iteration 400 / 1500: loss 2.990715
iteration 500 / 1500: loss 2.606107
iteration 600 / 1500: loss 2.949388
iteration 700 / 1500: loss 2.792148
iteration 800 / 1500: loss 2.788379
iteration 900 / 1500: loss 2.757446
iteration 1000 / 1500: loss 2.466590
iteration 1100 / 1500: loss 2.383806
iteration 1200 / 1500: loss 2.557642
iteration 1300 / 1500: loss 2.494052
iteration 1400 / 1500: loss 2.412724
Val_accuracy: 0.278, Current best_val: 0.4
count 443 / 600
iteration 0 / 20: loss 6.122544
iteration 0 / 1500: loss 4.962990
iteration 100 / 1500: loss 2.820245
iteration 200 / 1500: loss 2.194121
iteration 300 / 1500: loss 2.213110
iteration 400 / 1500: loss 2.112225
iteration 500 / 1500: loss 1.853569
iteration 600 / 1500: loss 2.073926
iteration 700 / 1500: loss 2.129187
iteration 800 / 1500: loss 1.961260
iteration 900 / 1500: loss 1.986010
iteration 1000 / 1500: loss 1.908766
iteration 1100 / 1500: loss 1.952621
iteration 1200 / 1500: loss 1.939858
iteration 1300 / 1500: loss 1.955139
iteration 1400 / 1500: loss 1.820214
Val_accuracy: 0.372, Current best_val: 0.4
count 444 / 600
iteration 0 / 20: loss 5.349497
iteration 0 / 1500: loss 5.838376
iteration 100 / 1500: loss 5.609864
iteration 200 / 1500: loss 5.857068
iteration 300 / 1500: loss 5.431997
iteration 400 / 1500: loss 5.377412
iteration 500 / 1500: loss 5.086540
iteration 600 / 1500: loss 5.413060
iteration 700 / 1500: loss 5.007403
iteration 800 / 1500: loss 4.982435
iteration 900 / 1500: loss 4.898281
iteration 1000 / 1500: loss 4.714866
iteration 1100 / 1500: loss 4.836304
iteration 1200 / 1500: loss 4.523573
iteration 1300 / 1500: loss 4.610319
iteration 1400 / 1500: loss 4.406688
Val_accuracy: 0.145, Current best_val: 0.4
count 445 / 600
iteration 0 / 20: loss 109.491063
count 446 / 600
iteration 0 / 20: loss 516.659770
count 447 / 600
iteration 0 / 20: loss 11.262444
iteration 0 / 1500: loss 9.650854
iteration 100 / 1500: loss 7.903504
iteration 200 / 1500: loss 6.985533
iteration 300 / 1500: loss 6.813284
iteration 400 / 1500: loss 6.864387
iteration 500 / 1500: loss 6.214485
iteration 600 / 1500: loss 6.422070
iteration 700 / 1500: loss 6.202311
iteration 800 / 1500: loss 5.871581
iteration 900 / 1500: loss 6.112487
iteration 1000 / 1500: loss 5.989849
iteration 1100 / 1500: loss 5.783938
iteration 1200 / 1500: loss 5.812562
iteration 1300 / 1500: loss 5.917031
iteration 1400 / 1500: loss 5.636797
```

```
Val_accuracy: 0.281, Current best_val: 0.4
count 448 / 600
iteration 0 / 20: loss 1254.775509
count 449 / 600
iteration 0 / 20: loss 1340.747242
count 450 / 600
iteration 0 / 20: loss 4.392332
iteration 0 / 1500: loss 4.835343
iteration 100 / 1500: loss 3.169329
iteration 200 / 1500: loss 2.945784
iteration 300 / 1500: loss 2.556183
iteration 400 / 1500: loss 2.376610
iteration 500 / 1500: loss 2.350438
iteration 600 / 1500: loss 2.373565
iteration 700 / 1500: loss 2.366345
iteration 800 / 1500: loss 2.325155
iteration 900 / 1500: loss 2.160256
iteration 1000 / 1500: loss 2.323293
iteration 1100 / 1500: loss 2.077052
iteration 1200 / 1500: loss 2.242662
iteration 1300 / 1500: loss 2.262505
iteration 1400 / 1500: loss 2.041994
Val_accuracy: 0.329, Current best_val: 0.4
count 451 / 600
iteration 0 / 20: loss 5.375442
iteration 0 / 1500: loss 6.433343
iteration 100 / 1500: loss 2.697280
iteration 200 / 1500: loss 2.375826
iteration 300 / 1500: loss 2.290922
iteration 400 / 1500: loss 2.070940
iteration 500 / 1500: loss 2.111032
iteration 600 / 1500: loss 1.962359
iteration 700 / 1500: loss 2.170176
iteration 800 / 1500: loss 2.017006
iteration 900 / 1500: loss 1.934423
iteration 1000 / 1500: loss 2.063765
iteration 1100 / 1500: loss 2.036266
iteration 1200 / 1500: loss 1.977606
iteration 1300 / 1500: loss 1.935645
iteration 1400 / 1500: loss 1.831683
Val_accuracy: 0.341, Current best_val: 0.4
count 452 / 600
iteration 0 / 20: loss 7.293075
iteration 0 / 1500: loss 6.978076
iteration 100 / 1500: loss 6.327255
iteration 200 / 1500: loss 6.509519
iteration 300 / 1500: loss 5.973436
iteration 400 / 1500: loss 5.784765
iteration 500 / 1500: loss 5.790883
iteration 600 / 1500: loss 5.527710
iteration 700 / 1500: loss 4.936706
iteration 800 / 1500: loss 4.950630
iteration 900 / 1500: loss 5.049571
iteration 1000 / 1500: loss 4.812529
iteration 1100 / 1500: loss 5.016396
iteration 1200 / 1500: loss 5.114100
iteration 1300 / 1500: loss 5.011427
iteration 1400 / 1500: loss 4.712109
Val_accuracy: 0.174, Current best_val: 0.4
count 453 / 600
iteration 0 / 20: loss 5.006962
iteration 0 / 1500: loss 5.690355
iteration 100 / 1500: loss 2.307292
iteration 200 / 1500: loss 2.294051
iteration 300 / 1500: loss 2.141028
iteration 400 / 1500: loss 2.152059
iteration 500 / 1500: loss 2.304467
iteration 600 / 1500: loss 2.236079
iteration 700 / 1500: loss 2.109209
iteration 800 / 1500: loss 1.944431
iteration 900 / 1500: loss 1.863763
iteration 1000 / 1500: loss 1.840468
iteration 1100 / 1500: loss 2.022386
iteration 1200 / 1500: loss 2.236031
iteration 1300 / 1500: loss 2.117732
iteration 1400 / 1500: loss 1.969621
Val_accuracy: 0.335, Current best_val: 0.4
count 454 / 600
iteration 0 / 20: loss 5.939310
iteration 0 / 1500: loss 7.469614
iteration 100 / 1500: loss 4.673982
iteration 200 / 1500: loss 4.421770
iteration 300 / 1500: loss 3.967015
iteration 400 / 1500: loss 3.494773
iteration 500 / 1500: loss 3.534835
iteration 600 / 1500: loss 3.630726
iteration 700 / 1500: loss 3.355402
iteration 800 / 1500: loss 3.385039
iteration 900 / 1500: loss 3.342620
iteration 1000 / 1500: loss 3.266272
iteration 1100 / 1500: loss 3.151602
iteration 1200 / 1500: loss 3.175276
iteration 1300 / 1500: loss 2.948314
iteration 1400 / 1500: loss 3.078872
Val_accuracy: 0.254, Current best_val: 0.4
count 455 / 600
```

```
iteration 0 / 20: loss 6.698960
iteration 0 / 1500: loss 6.223191
iteration 100 / 1500: loss 3.603113
iteration 200 / 1500: loss 3.532356
iteration 300 / 1500: loss 3.218954
iteration 400 / 1500: loss 2.998843
iteration 500 / 1500: loss 3.006650
iteration 600 / 1500: loss 3.023492
iteration 700 / 1500: loss 2.898000
iteration 800 / 1500: loss 2.927001
iteration 900 / 1500: loss 2.917955
iteration 1000 / 1500: loss 2.733981
iteration 1100 / 1500: loss 2.760682
iteration 1200 / 1500: loss 2.542677
iteration 1300 / 1500: loss 2.786835
iteration 1400 / 1500: loss 2.565665
Val_accuracy: 0.286, Current best_val: 0.4
count 456 / 600
iteration 0 / 20: loss 16.167634
count 457 / 600
iteration 0 / 20: loss 6.107823
iteration 0 / 1500: loss 6.579105
iteration 100 / 1500: loss 5.067533
iteration 200 / 1500: loss 4.548039
iteration 300 / 1500: loss 4.256534
iteration 400 / 1500: loss 3.875038
iteration 500 / 1500: loss 3.814025
iteration 600 / 1500: loss 3.582968
iteration 700 / 1500: loss 3.671966
iteration 800 / 1500: loss 3.568336
iteration 900 / 1500: loss 3.179800
iteration 1000 / 1500: loss 3.305781
iteration 1100 / 1500: loss 3.373311
iteration 1200 / 1500: loss 3.526411
iteration 1300 / 1500: loss 3.280276
iteration 1400 / 1500: loss 3.013776
Val_accuracy: 0.231, Current best_val: 0.4
count 458 / 600
iteration 0 / 20: loss 262.884870
count 459 / 600
iteration 0 / 20: loss 23.772425
count 460 / 600
iteration 0 / 20: loss 7.175497
iteration 0 / 1500: loss 5.441557
iteration 100 / 1500: loss 5.061433
iteration 200 / 1500: loss 5.753329
iteration 300 / 1500: loss 5.017970
iteration 400 / 1500: loss 5.283686
iteration 500 / 1500: loss 4.620565
iteration 600 / 1500: loss 4.996632
iteration 700 / 1500: loss 4.725500
iteration 800 / 1500: loss 4.681379
iteration 900 / 1500: loss 5.250051
iteration 1000 / 1500: loss 5.022511
iteration 1100 / 1500: loss 4.798189
iteration 1200 / 1500: loss 4.778209
iteration 1300 / 1500: loss 4.974241
iteration 1400 / 1500: loss 4.833889
Val_accuracy: 0.158, Current best_val: 0.4
count 461 / 600
iteration 0 / 20: loss 17.526846
count 462 / 600
iteration 0 / 20: loss 5.450394
iteration 0 / 1500: loss 5.246705
iteration 100 / 1500: loss 5.071863
iteration 200 / 1500: loss 4.987435
iteration 300 / 1500: loss 4.398932
iteration 400 / 1500: loss 4.674640
iteration 500 / 1500: loss 4.048728
iteration 600 / 1500: loss 4.671310
iteration 700 / 1500: loss 4.493814
iteration 800 / 1500: loss 4.236905
iteration 900 / 1500: loss 4.216809
iteration 1000 / 1500: loss 4.044397
iteration 1100 / 1500: loss 4.493508
iteration 1200 / 1500: loss 3.531276
iteration 1300 / 1500: loss 4.206078
iteration 1400 / 1500: loss 3.863816
Val_accuracy: 0.159, Current best_val: 0.4
count 463 / 600
iteration 0 / 20: loss 7.331213
iteration 0 / 1500: loss 6.559077
iteration 100 / 1500: loss 6.339830
iteration 200 / 1500: loss 6.408518
iteration 300 / 1500: loss 5.930455
iteration 400 / 1500: loss 5.502308
iteration 500 / 1500: loss 5.336202
iteration 600 / 1500: loss 5.585483
iteration 700 / 1500: loss 5.203684
iteration 800 / 1500: loss 5.402942
iteration 900 / 1500: loss 5.535563
iteration 1000 / 1500: loss 4.986790
iteration 1100 / 1500: loss 5.107871
iteration 1200 / 1500: loss 5.111778
iteration 1300 / 1500: loss 4.927067
iteration 1400 / 1500: loss 4.654170
```

```
Val_accuracy: 0.202, Current best_val: 0.4
count 464 / 600
iteration 0 / 20: loss 11.457361
iteration 0 / 1500: loss 11.321097
iteration 100 / 1500: loss 8.888266
iteration 200 / 1500: loss 8.663187
iteration 300 / 1500: loss 8.200757
iteration 400 / 1500: loss 8.169430
iteration 500 / 1500: loss 7.879488
iteration 600 / 1500: loss 7.664299
iteration 700 / 1500: loss 7.544089
iteration 800 / 1500: loss 7.244646
iteration 900 / 1500: loss 7.366167
iteration 1000 / 1500: loss 7.293696
iteration 1100 / 1500: loss 7.535805
iteration 1200 / 1500: loss 7.086279
iteration 1300 / 1500: loss 6.908375
iteration 1400 / 1500: loss 6.991878
Val_accuracy: 0.259, Current best_val: 0.4
count 465 / 600
iteration 0 / 20: loss 63.120945
count 466 / 600
iteration 0 / 20: loss 83.126877
count 467 / 600
iteration 0 / 20: loss 5.260284
iteration 0 / 1500: loss 5.391527
iteration 100 / 1500: loss 5.018314
iteration 200 / 1500: loss 4.221471
iteration 300 / 1500: loss 4.480203
iteration 400 / 1500: loss 3.904979
iteration 500 / 1500: loss 4.123726
iteration 600 / 1500: loss 3.999303
iteration 700 / 1500: loss 4.285054
iteration 800 / 1500: loss 4.022909
iteration 900 / 1500: loss 3.866468
iteration 1000 / 1500: loss 3.714103
iteration 1100 / 1500: loss 3.645664
iteration 1200 / 1500: loss 3.490261
iteration 1300 / 1500: loss 3.571682
iteration 1400 / 1500: loss 3.400766
Val_accuracy: 0.192, Current best_val: 0.4
count 468 / 600
iteration 0 / 20: loss 10.112070
iteration 0 / 1500: loss 10.962921
iteration 100 / 1500: loss 9.452518
iteration 200 / 1500: loss 8.833420
iteration 300 / 1500: loss 8.372131
iteration 400 / 1500: loss 8.178102
iteration 500 / 1500: loss 7.955977
iteration 600 / 1500: loss 7.805261
iteration 700 / 1500: loss 7.713657
iteration 800 / 1500: loss 7.840904
iteration 900 / 1500: loss 7.684667
iteration 1000 / 1500: loss 7.586123
iteration 1100 / 1500: loss 7.362178
iteration 1200 / 1500: loss 7.417583
iteration 1300 / 1500: loss 7.124183
iteration 1400 / 1500: loss 7.224542
Val_accuracy: 0.234, Current best_val: 0.4
count 469 / 600
iteration 0 / 20: loss 6.241358
iteration 0 / 1500: loss 6.463018
iteration 100 / 1500: loss 5.169654
iteration 200 / 1500: loss 4.870232
iteration 300 / 1500: loss 4.585337
iteration 400 / 1500: loss 3.914250
iteration 500 / 1500: loss 4.467776
iteration 600 / 1500: loss 3.966436
iteration 700 / 1500: loss 4.075195
iteration 800 / 1500: loss 3.854663
iteration 900 / 1500: loss 3.734393
iteration 1000 / 1500: loss 3.758387
iteration 1100 / 1500: loss 3.942613
iteration 1200 / 1500: loss 3.955925
iteration 1300 / 1500: loss 3.789299
iteration 1400 / 1500: loss 3.600525
Val_accuracy: 0.268, Current best_val: 0.4
count 470 / 600
iteration 0 / 20: loss 4.602855
iteration 0 / 1500: loss 6.189886
iteration 100 / 1500: loss 3.225042
iteration 200 / 1500: loss 2.850374
iteration 300 / 1500: loss 2.436090
iteration 400 / 1500: loss 2.606765
iteration 500 / 1500: loss 2.314466
iteration 600 / 1500: loss 2.427675
iteration 700 / 1500: loss 2.215476
iteration 800 / 1500: loss 2.300250
iteration 900 / 1500: loss 2.263744
iteration 1000 / 1500: loss 2.148116
iteration 1100 / 1500: loss 2.195986
iteration 1200 / 1500: loss 2.329929
iteration 1300 / 1500: loss 2.256644
iteration 1400 / 1500: loss 1.975868
Val_accuracy: 0.348, Current best_val: 0.4
count 471 / 600
```

```
iteration 0 / 20: loss 4.854001
iteration 0 / 1500: loss 5.525615
iteration 100 / 1500: loss 5.803325
iteration 200 / 1500: loss 5.707416
iteration 300 / 1500: loss 5.493176
iteration 400 / 1500: loss 5.248948
iteration 500 / 1500: loss 5.013812
iteration 600 / 1500: loss 4.937337
iteration 700 / 1500: loss 4.853124
iteration 800 / 1500: loss 4.731455
iteration 900 / 1500: loss 4.868760
iteration 1000 / 1500: loss 5.027519
iteration 1100 / 1500: loss 4.977210
iteration 1200 / 1500: loss 4.860760
iteration 1300 / 1500: loss 4.445487
iteration 1400 / 1500: loss 4.330039
Val_accuracy: 0.134, Current best_val: 0.4
count 472 / 600
iteration 0 / 20: loss 5.836713
iteration 0 / 1500: loss 6.746972
iteration 100 / 1500: loss 2.813048
iteration 200 / 1500: loss 2.634225
iteration 300 / 1500: loss 2.518301
iteration 400 / 1500: loss 2.333650
iteration 500 / 1500: loss 2.387227
iteration 600 / 1500: loss 2.339156
iteration 700 / 1500: loss 2.398234
iteration 800 / 1500: loss 2.399951
iteration 900 / 1500: loss 2.256030
iteration 1000 / 1500: loss 2.151130
iteration 1100 / 1500: loss 2.219936
iteration 1200 / 1500: loss 2.209757
iteration 1300 / 1500: loss 2.119308
iteration 1400 / 1500: loss 2.089665
Val_accuracy: 0.374, Current best_val: 0.4
count 473 / 600
iteration 0 / 20: loss 7.238595
iteration 0 / 1500: loss 6.680187
iteration 100 / 1500: loss 4.497713
iteration 200 / 1500: loss 4.214314
iteration 300 / 1500: loss 4.050232
iteration 400 / 1500: loss 4.060602
iteration 500 / 1500: loss 3.923866
iteration 600 / 1500: loss 3.812183
iteration 700 / 1500: loss 3.387702
iteration 800 / 1500: loss 2.988492
iteration 900 / 1500: loss 3.262875
iteration 1000 / 1500: loss 3.102786
iteration 1100 / 1500: loss 3.449199
iteration 1200 / 1500: loss 2.988815
iteration 1300 / 1500: loss 3.080039
iteration 1400 / 1500: loss 3.370843
Val_accuracy: 0.258, Current best_val: 0.4
count 474 / 600
iteration 0 / 20: loss 504.004570
count 475 / 600
iteration 0 / 20: loss 2868.547785
count 476 / 600
iteration 0 / 20: loss 488.280281
count 477 / 600
iteration 0 / 20: loss 1149.373021
count 478 / 600
iteration 0 / 20: loss 78.554777
count 479 / 600
iteration 0 / 20: loss 400.717958
count 480 / 600
iteration 0 / 20: loss 6.703930
iteration 0 / 1500: loss 5.791656
iteration 100 / 1500: loss 4.432350
iteration 200 / 1500: loss 3.858497
iteration 300 / 1500: loss 3.493816
iteration 400 / 1500: loss 3.728290
iteration 500 / 1500: loss 3.298731
iteration 600 / 1500: loss 3.295225
iteration 700 / 1500: loss 3.054322
iteration 800 / 1500: loss 3.098048
iteration 900 / 1500: loss 3.199353
iteration 1000 / 1500: loss 2.949511
iteration 1100 / 1500: loss 2.882197
iteration 1200 / 1500: loss 3.190873
iteration 1300 / 1500: loss 2.753967
iteration 1400 / 1500: loss 2.900757
Val_accuracy: 0.258, Current best_val: 0.4
count 481 / 600
iteration 0 / 20: loss 138.451016
count 482 / 600
iteration 0 / 20: loss 9.529413
iteration 0 / 1500: loss 10.123170
iteration 100 / 1500: loss 8.008392
iteration 200 / 1500: loss 7.664637
iteration 300 / 1500: loss 7.569384
iteration 400 / 1500: loss 7.142697
iteration 500 / 1500: loss 6.849798
iteration 600 / 1500: loss 6.858400
iteration 700 / 1500: loss 6.840647
iteration 800 / 1500: loss 6.609855
iteration 900 / 1500: loss 6.449850
iteration 1000 / 1500: loss 6.291466
iteration 1100 / 1500: loss 6.340085
iteration 1200 / 1500: loss 6.155569
iteration 1300 / 1500: loss 6.172150
iteration 1400 / 1500: loss 5.886783
Val_accuracy: 0.298, Current best_val: 0.4
count 483 / 600
iteration 0 / 20: loss 5.103030
iteration 0 / 1500: loss 5.605320
iteration 100 / 1500: loss 4.298050
iteration 200 / 1500: loss 3.657794
iteration 300 / 1500: loss 3.350133
iteration 400 / 1500: loss 3.164275
iteration 500 / 1500: loss 3.169273
iteration 600 / 1500: loss 3.423992
iteration 700 / 1500: loss 2.903899
iteration 800 / 1500: loss 3.022642
iteration 900 / 1500: loss 2.692460
iteration 1000 / 1500: loss 2.837553
iteration 1100 / 1500: loss 2.834729
iteration 1200 / 1500: loss 2.811942
iteration 1300 / 1500: loss 2.816294
iteration 1400 / 1500: loss 2.705753
Val_accuracy: 0.235, Current best_val: 0.4
count 484 / 600
iteration 0 / 20: loss 6.106140
iteration 0 / 1500: loss 5.791793
iteration 100 / 1500: loss 5.598259
iteration 200 / 1500: loss 5.312080
iteration 300 / 1500: loss 5.661913
iteration 400 / 1500: loss 5.446636
iteration 500 / 1500: loss 5.145435
iteration 600 / 1500: loss 5.172667
iteration 700 / 1500: loss 5.292519
iteration 800 / 1500: loss 5.024093
iteration 900 / 1500: loss 4.991230
iteration 1000 / 1500: loss 4.973041
iteration 1100 / 1500: loss 4.774617
iteration 1200 / 1500: loss 4.652722
iteration 1300 / 1500: loss 5.012619
iteration 1400 / 1500: loss 4.634099
Val_accuracy: 0.116, Current best_val: 0.4
count 485 / 600
iteration 0 / 20: loss 126.718797
count 486 / 600
iteration 0 / 20: loss 5.048242
iteration 0 / 1500: loss 6.295015
iteration 100 / 1500: loss 5.887277
iteration 200 / 1500: loss 5.793105
iteration 300 / 1500: loss 6.104052
iteration 400 / 1500: loss 5.840321
iteration 500 / 1500: loss 5.856875
iteration 600 / 1500: loss 5.118479
iteration 700 / 1500: loss 5.425571
iteration 800 / 1500: loss 5.131874
iteration 900 / 1500: loss 4.832222
iteration 1000 / 1500: loss 5.247010
iteration 1100 / 1500: loss 5.051493
iteration 1200 / 1500: loss 5.267947
iteration 1300 / 1500: loss 5.046357
iteration 1400 / 1500: loss 5.307208
Val_accuracy: 0.126, Current best_val: 0.4
count 487 / 600
iteration 0 / 20: loss 5.429650
iteration 0 / 1500: loss 5.824796
iteration 100 / 1500: loss 3.298228
iteration 200 / 1500: loss 2.821609
iteration 300 / 1500: loss 2.689988
iteration 400 / 1500: loss 2.454021
iteration 500 / 1500: loss 2.576748
iteration 600 / 1500: loss 2.593007
iteration 700 / 1500: loss 2.478247
iteration 800 / 1500: loss 2.391247
iteration 900 / 1500: loss 2.526061
iteration 1000 / 1500: loss 2.442324
iteration 1100 / 1500: loss 2.406567
iteration 1200 / 1500: loss 2.450213
iteration 1300 / 1500: loss 2.297344
iteration 1400 / 1500: loss 2.239085
Val_accuracy: 0.288, Current best_val: 0.4
count 488 / 600
iteration 0 / 20: loss 38.090751
count 489 / 600
iteration 0 / 20: loss 36.445511
count 490 / 600
iteration 0 / 20: loss 38.291427
count 491 / 600
iteration 0 / 20: loss 4.550142
iteration 0 / 1500: loss 5.623613
iteration 100 / 1500: loss 2.933226
iteration 200 / 1500: loss 2.596732
iteration 300 / 1500: loss 2.429449
iteration 400 / 1500: loss 2.310846
iteration 500 / 1500: loss 2.359985
iteration 600 / 1500: loss 2.315616
iteration 700 / 1500: loss 2.303421
iteration 800 / 1500: loss 2.052192
iteration 900 / 1500: loss 2.344360
iteration 1000 / 1500: loss 2.224024
iteration 1100 / 1500: loss 1.945451
iteration 1200 / 1500: loss 2.179600
iteration 1300 / 1500: loss 2.108941
iteration 1400 / 1500: loss 2.269655
Val_accuracy: 0.315, Current best_val: 0.4
count 492 / 600
iteration 0 / 20: loss 1151.287165
count 493 / 600
iteration 0 / 20: loss 2184.577089
count 494 / 600
iteration 0 / 20: loss 5.731366
iteration 0 / 1500: loss 5.715206
iteration 100 / 1500: loss 6.719829
iteration 200 / 1500: loss 6.785987
iteration 300 / 1500: loss 6.906491
iteration 400 / 1500: loss 6.389552
iteration 500 / 1500: loss 5.796487
iteration 600 / 1500: loss 6.517041
iteration 700 / 1500: loss 5.785663
iteration 800 / 1500: loss 5.681054
iteration 900 / 1500: loss 5.861544
iteration 1000 / 1500: loss 6.205253
iteration 1100 / 1500: loss 5.709007
iteration 1200 / 1500: loss 5.380273
iteration 1300 / 1500: loss 5.253249
iteration 1400 / 1500: loss 5.761253
Val_accuracy: 0.095, Current best_val: 0.4
count 495 / 600
iteration 0 / 20: loss 502.442312
count 496 / 600
iteration 0 / 20: loss 6.964350
iteration 0 / 1500: loss 6.234960
iteration 100 / 1500: loss 6.718831
iteration 200 / 1500: loss 6.359730
iteration 300 / 1500: loss 6.366850
iteration 400 / 1500: loss 6.434535
iteration 500 / 1500: loss 6.284289
iteration 600 / 1500: loss 6.282251
iteration 700 / 1500: loss 5.972224
iteration 800 / 1500: loss 6.327371
iteration 900 / 1500: loss 5.794356
iteration 1000 / 1500: loss 5.945721
iteration 1100 / 1500: loss 6.093587
iteration 1200 / 1500: loss 6.024221
iteration 1300 / 1500: loss 6.150590
iteration 1400 / 1500: loss 6.145551
Val_accuracy: 0.091, Current best_val: 0.4
count 497 / 600
iteration 0 / 20: loss 22.140757
count 498 / 600
iteration 0 / 20: loss 5.830990
iteration 0 / 1500: loss 5.251706
iteration 100 / 1500: loss 4.593752
iteration 200 / 1500: loss 4.460011
iteration 300 / 1500: loss 4.707982
iteration 400 / 1500: loss 5.061358
iteration 500 / 1500: loss 4.294741
iteration 600 / 1500: loss 4.465677
iteration 700 / 1500: loss 4.182962
iteration 800 / 1500: loss 4.479262
iteration 900 / 1500: loss 4.228436
iteration 1000 / 1500: loss 4.392352
iteration 1100 / 1500: loss 3.903783
iteration 1200 / 1500: loss 4.114635
iteration 1300 / 1500: loss 3.992752
iteration 1400 / 1500: loss 3.903245
Val_accuracy: 0.179, Current best_val: 0.4
count 499 / 600
iteration 0 / 20: loss 362.258956
count 500 / 600
iteration 0 / 20: loss 7.346387
iteration 0 / 1500: loss 8.002861
iteration 100 / 1500: loss 5.330919
iteration 200 / 1500: loss 4.898609
iteration 300 / 1500: loss 4.499747
iteration 400 / 1500: loss 4.347171
iteration 500 / 1500: loss 4.069403
iteration 600 / 1500: loss 4.168674
iteration 700 / 1500: loss 3.953520
iteration 800 / 1500: loss 4.223624
iteration 900 / 1500: loss 4.159505
iteration 1000 / 1500: loss 4.036197
iteration 1100 / 1500: loss 4.051618
iteration 1200 / 1500: loss 3.787407
iteration 1300 / 1500: loss 3.967759
iteration 1400 / 1500: loss 3.555242
Val_accuracy: 0.284, Current best_val: 0.4
count 501 / 600
iteration 0 / 20: loss 5.897910
iteration 0 / 1500: loss 6.436090
iteration 100 / 1500: loss 6.209152
iteration 200 / 1500: loss 5.316066
iteration 300 / 1500: loss 4.928596
iteration 400 / 1500: loss 5.361975
iteration 500 / 1500: loss 5.155909
iteration 600 / 1500: loss 4.661661
iteration 700 / 1500: loss 4.688625
iteration 800 / 1500: loss 4.331678
iteration 900 / 1500: loss 4.089935
iteration 1000 / 1500: loss 4.191640
iteration 1100 / 1500: loss 4.332904
iteration 1200 / 1500: loss 4.286385
iteration 1300 / 1500: loss 4.026210
iteration 1400 / 1500: loss 3.990704
Val_accuracy: 0.134, Current best_val: 0.4
count 502 / 600
iteration 0 / 20: loss 14.691555
iteration 0 / 1500: loss 15.704014
iteration 100 / 1500: loss 8.305420
iteration 200 / 1500: loss 5.657637
iteration 300 / 1500: loss 4.420247
iteration 400 / 1500: loss 3.485070
iteration 500 / 1500: loss 2.873458
iteration 600 / 1500: loss 2.605411
iteration 700 / 1500: loss 2.181412
iteration 800 / 1500: loss 2.065020
iteration 900 / 1500: loss 2.087664
iteration 1000 / 1500: loss 1.925455
iteration 1100 / 1500: loss 1.937266
iteration 1200 / 1500: loss 1.881379
iteration 1300 / 1500: loss 1.846511
iteration 1400 / 1500: loss 1.841216
Val_accuracy: 0.389, Current best_val: 0.4
count 503 / 600
iteration 0 / 20: loss 19.970035
count 504 / 600
iteration 0 / 20: loss 5.741324
iteration 0 / 1500: loss 5.586328
iteration 100 / 1500: loss 4.204200
iteration 200 / 1500: loss 3.832009
iteration 300 / 1500: loss 3.634008
iteration 400 / 1500: loss 3.532103
iteration 500 / 1500: loss 3.201412
iteration 600 / 1500: loss 3.324765
iteration 700 / 1500: loss 3.287994
iteration 800 / 1500: loss 3.089661
iteration 900 / 1500: loss 3.201162
iteration 1000 / 1500: loss 3.274279
iteration 1100 / 1500: loss 2.828050
iteration 1200 / 1500: loss 2.877537
iteration 1300 / 1500: loss 2.942830
iteration 1400 / 1500: loss 2.828248
Val_accuracy: 0.223, Current best_val: 0.4
count 505 / 600
iteration 0 / 20: loss 6.038925
iteration 0 / 1500: loss 5.229197
iteration 100 / 1500: loss 4.682535
iteration 200 / 1500: loss 4.540202
iteration 300 / 1500: loss 4.543236
iteration 400 / 1500: loss 4.445517
iteration 500 / 1500: loss 4.106707
iteration 600 / 1500: loss 3.862884
iteration 700 / 1500: loss 4.122359
iteration 800 / 1500: loss 3.585353
iteration 900 / 1500: loss 3.558028
iteration 1000 / 1500: loss 3.736944
iteration 1100 / 1500: loss 3.671584
iteration 1200 / 1500: loss 3.377158
iteration 1300 / 1500: loss 3.735878
iteration 1400 / 1500: loss 3.361103
Val_accuracy: 0.191, Current best_val: 0.4
count 506 / 600
iteration 0 / 20: loss 6.019067
iteration 0 / 1500: loss 6.589221
iteration 100 / 1500: loss 6.146706
iteration 200 / 1500: loss 6.554195
iteration 300 / 1500: loss 6.017162
iteration 400 / 1500: loss 6.164297
iteration 500 / 1500: loss 5.782601
iteration 600 / 1500: loss 5.985024
iteration 700 / 1500: loss 6.004172
iteration 800 / 1500: loss 5.727553
iteration 900 / 1500: loss 5.457412
iteration 1000 / 1500: loss 5.188281
iteration 1100 / 1500: loss 5.721215
iteration 1200 / 1500: loss 5.166102
iteration 1300 / 1500: loss 5.482615
iteration 1400 / 1500: loss 5.496358
Val_accuracy: 0.098, Current best_val: 0.4
count 507 / 600
iteration 0 / 20: loss 502.570704
count 508 / 600
iteration 0 / 20: loss 21.222302
count 509 / 600
iteration 0 / 20: loss 34.697990
count 510 / 600
iteration 0 / 20: loss 6.824290
iteration 0 / 1500: loss 6.822941
iteration 100 / 1500: loss 5.076483
iteration 200 / 1500: loss 4.771188
iteration 300 / 1500: loss 4.416678
iteration 400 / 1500: loss 4.480572
iteration 500 / 1500: loss 4.217156
iteration 600 / 1500: loss 4.347881
iteration 700 / 1500: loss 4.366902
iteration 800 / 1500: loss 4.034940
iteration 900 / 1500: loss 4.085574
iteration 1000 / 1500: loss 4.167538
iteration 1100 / 1500: loss 4.028359
iteration 1200 / 1500: loss 3.917513
iteration 1300 / 1500: loss 4.150861
iteration 1400 / 1500: loss 3.756508
Val_accuracy: 0.252, Current best_val: 0.4
count 511 / 600
iteration 0 / 20: loss 5.278017
iteration 0 / 1500: loss 6.259285
iteration 100 / 1500: loss 2.431524
iteration 200 / 1500: loss 1.978882
iteration 300 / 1500: loss 2.179980
iteration 400 / 1500: loss 2.014768
iteration 500 / 1500: loss 2.001332
iteration 600 / 1500: loss 1.838441
iteration 700 / 1500: loss 1.769926
iteration 800 / 1500: loss 2.015735
iteration 900 / 1500: loss 1.759900
iteration 1000 / 1500: loss 1.970632
iteration 1100 / 1500: loss 1.783798
iteration 1200 / 1500: loss 1.759889
iteration 1300 / 1500: loss 1.959099
iteration 1400 / 1500: loss 1.785368
Val_accuracy: 0.353, Current best_val: 0.4
count 512 / 600
iteration 0 / 20: loss 211.238888
count 513 / 600
iteration 0 / 20: loss 193.013211
count 514 / 600
iteration 0 / 20: loss 6.064484
iteration 0 / 1500: loss 6.057177
iteration 100 / 1500: loss 3.590213
iteration 200 / 1500: loss 2.804128
iteration 300 / 1500: loss 2.998116
iteration 400 / 1500: loss 2.450794
iteration 500 / 1500: loss 2.510115
iteration 600 / 1500: loss 2.609298
iteration 700 / 1500: loss 2.475303
iteration 800 / 1500: loss 2.259731
iteration 900 / 1500: loss 2.270690
iteration 1000 / 1500: loss 2.216931
iteration 1100 / 1500: loss 2.417593
iteration 1200 / 1500: loss 2.445384
iteration 1300 / 1500: loss 2.364961
iteration 1400 / 1500: loss 2.112054
Val_accuracy: 0.295, Current best_val: 0.4
count 515 / 600
iteration 0 / 20: loss 21.535406
count 516 / 600
iteration 0 / 20: loss 6.426043
iteration 0 / 1500: loss 5.697649
iteration 100 / 1500: loss 4.195648
iteration 200 / 1500: loss 3.677317
iteration 300 / 1500: loss 3.431724
iteration 400 / 1500: loss 3.341242
iteration 500 / 1500: loss 3.228877
iteration 600 / 1500: loss 3.033331
iteration 700 / 1500: loss 2.677806
iteration 800 / 1500: loss 2.726933
iteration 900 / 1500: loss 2.621053
iteration 1000 / 1500: loss 2.978389
iteration 1100 / 1500: loss 3.062501
iteration 1200 / 1500: loss 2.846845
iteration 1300 / 1500: loss 2.494502
iteration 1400 / 1500: loss 2.573170
Val_accuracy: 0.256, Current best_val: 0.4
count 517 / 600
iteration 0 / 20: loss 14.663898
iteration 0 / 1500: loss 13.436745
iteration 100 / 1500: loss 10.040622
iteration 200 / 1500: loss 9.120008
iteration 300 / 1500: loss 8.233627
iteration 400 / 1500: loss 7.416731
iteration 500 / 1500: loss 6.856038
iteration 600 / 1500: loss 6.443347
iteration 700 / 1500: loss 5.878389
iteration 800 / 1500: loss 5.402213
iteration 900 / 1500: loss 5.127045
iteration 1000 / 1500: loss 4.727376
iteration 1100 / 1500: loss 4.458971
iteration 1200 / 1500: loss 4.251029
iteration 1300 / 1500: loss 4.033096
iteration 1400 / 1500: loss 3.997349
Val_accuracy: 0.384, Current best_val: 0.4
count 518 / 600
iteration 0 / 20: loss 2468.897659
count 519 / 600
```

```
iteration 0 / 20: loss 6.104847
iteration 0 / 1500: loss 5.474754
iteration 100 / 1500: loss 4.196725
iteration 200 / 1500: loss 3.995482
iteration 300 / 1500: loss 3.462143
iteration 400 / 1500: loss 3.476168
iteration 500 / 1500: loss 3.386665
iteration 600 / 1500: loss 3.317534
iteration 700 / 1500: loss 3.021616
iteration 800 / 1500: loss 2.804598
iteration 900 / 1500: loss 2.990650
iteration 1000 / 1500: loss 3.108763
iteration 1100 / 1500: loss 2.711507
iteration 1200 / 1500: loss 2.680945
iteration 1300 / 1500: loss 2.645011
iteration 1400 / 1500: loss 2.657182
Val_accuracy: 0.227, Current best_val: 0.4
count 520 / 600
iteration 0 / 20: loss 5.888126
iteration 0 / 1500: loss 6.272666
iteration 100 / 1500: loss 3.038078
iteration 200 / 1500: loss 2.912796
iteration 300 / 1500: loss 2.566768
iteration 400 / 1500: loss 2.578446
iteration 500 / 1500: loss 2.546502
iteration 600 / 1500: loss 2.191728
iteration 700 / 1500: loss 2.193520
iteration 800 / 1500: loss 2.266467
iteration 900 / 1500: loss 2.356201
iteration 1000 / 1500: loss 2.409755
iteration 1100 / 1500: loss 2.294850
iteration 1200 / 1500: loss 2.237452
iteration 1300 / 1500: loss 2.104379
iteration 1400 / 1500: loss 2.183286
Val_accuracy: 0.306, Current best_val: 0.4
count 521 / 600
iteration 0 / 20: loss 6.267120
iteration 0 / 1500: loss 5.564997
iteration 100 / 1500: loss 5.199862
iteration 200 / 1500: loss 5.679812
iteration 300 / 1500: loss 5.182742
iteration 400 / 1500: loss 5.279208
iteration 500 / 1500: loss 5.235393
iteration 600 / 1500: loss 4.816986
iteration 700 / 1500: loss 4.985292
iteration 800 / 1500: loss 4.646752
iteration 900 / 1500: loss 4.914086
iteration 1000 / 1500: loss 4.693739
iteration 1100 / 1500: loss 4.718336
iteration 1200 / 1500: loss 4.956929
iteration 1300 / 1500: loss 4.966776
iteration 1400 / 1500: loss 4.735729
Val_accuracy: 0.087, Current best_val: 0.4
count 522 / 600
iteration 0 / 20: loss 545.609778
count 523 / 600
iteration 0 / 20: loss 5.745754
iteration 0 / 1500: loss 5.476870
iteration 100 / 1500: loss 5.526794
iteration 200 / 1500: loss 5.477571
iteration 300 / 1500: loss 5.567225
iteration 400 / 1500: loss 5.900003
iteration 500 / 1500: loss 5.557666
iteration 600 / 1500: loss 6.081193
iteration 700 / 1500: loss 5.252219
iteration 800 / 1500: loss 5.760948
iteration 900 / 1500: loss 5.436730
iteration 1000 / 1500: loss 5.395518
iteration 1100 / 1500: loss 5.594453
iteration 1200 / 1500: loss 5.693730
iteration 1300 / 1500: loss 5.405798
iteration 1400 / 1500: loss 5.243098
Val_accuracy: 0.104, Current best_val: 0.4
count 524 / 600
iteration 0 / 20: loss 821.715986
iteration 0 / 1500: loss 818.890262
iteration 100 / 1500: loss 2.878370
iteration 200 / 1500: loss 2.561102
iteration 300 / 1500: loss 3.060583
iteration 400 / 1500: loss 2.828052
iteration 500 / 1500: loss 3.009097
iteration 600 / 1500: loss 2.888285
iteration 700 / 1500: loss 2.822848
iteration 800 / 1500: loss 2.764134
iteration 900 / 1500: loss 3.031144
iteration 1000 / 1500: loss 2.228534
iteration 1100 / 1500: loss 2.959343
iteration 1200 / 1500: loss 3.755244
iteration 1300 / 1500: loss 3.041406
iteration 1400 / 1500: loss 2.938926
Val_accuracy: 0.202, Current best_val: 0.4
count 525 / 600
iteration 0 / 20: loss 6.288810
iteration 0 / 1500: loss 5.693256
iteration 100 / 1500: loss 6.060789
iteration 200 / 1500: loss 5.216967
iteration 300 / 1500: loss 4.959769
iteration 400 / 1500: loss 5.120907
iteration 500 / 1500: loss 4.931101
iteration 600 / 1500: loss 4.919242
iteration 700 / 1500: loss 4.915672
iteration 800 / 1500: loss 4.457879
iteration 900 / 1500: loss 4.421842
iteration 1000 / 1500: loss 4.259615
iteration 1100 / 1500: loss 4.299140
iteration 1200 / 1500: loss 4.350803
iteration 1300 / 1500: loss 3.991901
iteration 1400 / 1500: loss 4.144881
Val_accuracy: 0.118, Current best_val: 0.4
count 526 / 600
iteration 0 / 20: loss 10.597705
iteration 0 / 1500: loss 10.915726
iteration 100 / 1500: loss 7.913200
iteration 200 / 1500: loss 7.308912
iteration 300 / 1500: loss 7.266793
iteration 400 / 1500: loss 7.080343
iteration 500 / 1500: loss 6.712710
iteration 600 / 1500: loss 6.524086
iteration 700 / 1500: loss 6.333198
iteration 800 / 1500: loss 6.171205
iteration 900 / 1500: loss 6.046248
iteration 1000 / 1500: loss 5.909805
iteration 1100 / 1500: loss 6.089943
iteration 1200 / 1500: loss 5.758009
iteration 1300 / 1500: loss 5.613232
iteration 1400 / 1500: loss 5.510966
Val_accuracy: 0.328, Current best_val: 0.4
count 527 / 600
iteration 0 / 20: loss 32.573077
count 528 / 600
iteration 0 / 20: loss 5.257863
iteration 0 / 1500: loss 6.007816
iteration 100 / 1500: loss 2.332143
iteration 200 / 1500: loss 2.381735
iteration 300 / 1500: loss 2.227937
iteration 400 / 1500: loss 2.239203
iteration 500 / 1500: loss 2.722265
iteration 600 / 1500: loss 2.454317
iteration 700 / 1500: loss 2.358867
iteration 800 / 1500: loss 2.023687
iteration 900 / 1500: loss 2.342399
iteration 1000 / 1500: loss 2.014290
iteration 1100 / 1500: loss 2.805141
iteration 1200 / 1500: loss 2.181875
iteration 1300 / 1500: loss 2.710370
iteration 1400 / 1500: loss 3.132569
Val_accuracy: 0.34, Current best_val: 0.4
count 529 / 600
iteration 0 / 20: loss 6.976527
iteration 0 / 1500: loss 6.393717
iteration 100 / 1500: loss 5.154239
iteration 200 / 1500: loss 4.484845
iteration 300 / 1500: loss 4.000810
iteration 400 / 1500: loss 4.217520
iteration 500 / 1500: loss 4.009895
iteration 600 / 1500: loss 4.029932
iteration 700 / 1500: loss 3.734461
iteration 800 / 1500: loss 3.429044
iteration 900 / 1500: loss 4.023636
iteration 1000 / 1500: loss 3.657801
iteration 1100 / 1500: loss 3.546123
iteration 1200 / 1500: loss 3.524061
iteration 1300 / 1500: loss 3.513221
iteration 1400 / 1500: loss 3.589455
Val_accuracy: 0.238, Current best_val: 0.4
count 530 / 600
iteration 0 / 20: loss 16.325872
count 531 / 600
iteration 0 / 20: loss 5.474963
iteration 0 / 1500: loss 5.701008
iteration 100 / 1500: loss 2.552799
iteration 200 / 1500: loss 1.991480
iteration 300 / 1500: loss 2.074964
iteration 400 / 1500: loss 2.080096
iteration 500 / 1500: loss 1.872057
iteration 600 / 1500: loss 1.937989
iteration 700 / 1500: loss 2.045861
iteration 800 / 1500: loss 1.786873
iteration 900 / 1500: loss 1.921720
iteration 1000 / 1500: loss 1.854410
iteration 1100 / 1500: loss 1.847829
iteration 1200 / 1500: loss 1.966698
iteration 1300 / 1500: loss 1.902203
iteration 1400 / 1500: loss 1.849326
Val_accuracy: 0.371, Current best_val: 0.4
count 532 / 600
iteration 0 / 20: loss 5.796614
iteration 0 / 1500: loss 5.526458
iteration 100 / 1500: loss 3.221702
iteration 200 / 1500: loss 2.675808
iteration 300 / 1500: loss 2.544229
iteration 400 / 1500: loss 2.548317
iteration 500 / 1500: loss 2.295904
iteration 600 / 1500: loss 2.440822
iteration 700 / 1500: loss 2.217873
iteration 800 / 1500: loss 2.355349
iteration 900 / 1500: loss 2.334240
iteration 1000 / 1500: loss 2.255437
iteration 1100 / 1500: loss 2.108364
iteration 1200 / 1500: loss 2.207738
iteration 1300 / 1500: loss 2.127907
iteration 1400 / 1500: loss 2.063870
Val_accuracy: 0.301, Current best_val: 0.4
count 533 / 600
iteration 0 / 20: loss 562.719438
count 534 / 600
iteration 0 / 20: loss 6.688581
iteration 0 / 1500: loss 6.431833
iteration 100 / 1500: loss 5.942813
iteration 200 / 1500: loss 5.759341
iteration 300 / 1500: loss 5.636397
iteration 400 / 1500: loss 5.840955
iteration 500 / 1500: loss 5.432072
iteration 600 / 1500: loss 5.718196
iteration 700 / 1500: loss 5.273946
iteration 800 / 1500: loss 5.072260
iteration 900 / 1500: loss 4.821672
iteration 1000 / 1500: loss 5.109550
iteration 1100 / 1500: loss 4.896337
iteration 1200 / 1500: loss 4.796469
iteration 1300 / 1500: loss 5.077732
iteration 1400 / 1500: loss 4.812083
Val_accuracy: 0.142, Current best_val: 0.4
count 535 / 600
iteration 0 / 20: loss 7.974849
iteration 0 / 1500: loss 6.454596
iteration 100 / 1500: loss 3.487485
iteration 200 / 1500: loss 2.996475
iteration 300 / 1500: loss 2.963963
iteration 400 / 1500: loss 2.870538
iteration 500 / 1500: loss 2.791671
iteration 600 / 1500: loss 2.579943
iteration 700 / 1500: loss 2.581041
iteration 800 / 1500: loss 2.321490
iteration 900 / 1500: loss 2.411385
iteration 1000 / 1500: loss 2.310862
iteration 1100 / 1500: loss 2.200144
iteration 1200 / 1500: loss 2.109200
iteration 1300 / 1500: loss 2.373069
iteration 1400 / 1500: loss 2.274033
```

```
Val_accuracy: 0.371, Current best_val: 0.4
count 536 / 600
iteration 0 / 20: loss 5.987756
iteration 0 / 1500: loss 5.869716
iteration 100 / 1500: loss 2.670223
iteration 200 / 1500: loss 2.858358
iteration 300 / 1500: loss 3.133559
iteration 400 / 1500: loss 2.445783
iteration 500 / 1500: loss 2.998566
iteration 600 / 1500: loss 2.643561
iteration 700 / 1500: loss 2.629271
iteration 800 / 1500: loss 2.357587
iteration 900 / 1500: loss 2.601325
iteration 1000 / 1500: loss 4.016067
iteration 1100 / 1500: loss 3.286666
iteration 1200 / 1500: loss 1.933643
iteration 1300 / 1500: loss 2.415124
iteration 1400 / 1500: loss 2.487739
Val_accuracy: 0.346, Current best_val: 0.4
count 537 / 600
iteration 0 / 20: loss 5.793603
iteration 0 / 1500: loss 5.385221
iteration 100 / 1500: loss 6.186584
iteration 200 / 1500: loss 5.829443
iteration 300 / 1500: loss 5.766341
iteration 400 / 1500: loss 5.609156
iteration 500 / 1500: loss 5.930054
iteration 600 / 1500: loss 5.393883
iteration 700 / 1500: loss 4.943072
iteration 800 / 1500: loss 5.551433
iteration 900 / 1500: loss 5.711419
iteration 1000 / 1500: loss 5.668298
iteration 1100 / 1500: loss 5.250550
iteration 1200 / 1500: loss 5.052231
iteration 1300 / 1500: loss 5.400964
iteration 1400 / 1500: loss 5.013020
Val_accuracy: 0.123, Current best_val: 0.4
count 538 / 600
iteration 0 / 20: loss 5.404576
iteration 0 / 1500: loss 6.245385
iteration 100 / 1500: loss 5.046349
iteration 200 / 1500: loss 4.476212
iteration 300 / 1500: loss 4.179072
iteration 400 / 1500: loss 3.802648
iteration 500 / 1500: loss 3.778054
iteration 600 / 1500: loss 3.629933
iteration 700 / 1500: loss 3.649867
iteration 800 / 1500: loss 3.251283
iteration 900 / 1500: loss 3.486830
iteration 1000 / 1500: loss 3.341604
iteration 1100 / 1500: loss 3.273679
iteration 1200 / 1500: loss 3.190074
iteration 1300 / 1500: loss 3.367641
iteration 1400 / 1500: loss 3.352964
Val_accuracy: 0.191, Current best_val: 0.4
count 539 / 600
iteration 0 / 20: loss 5.323150
iteration 0 / 1500: loss 5.369924
iteration 100 / 1500: loss 5.195507
iteration 200 / 1500: loss 4.778034
iteration 300 / 1500: loss 5.043126
iteration 400 / 1500: loss 5.035002
iteration 500 / 1500: loss 5.384664
iteration 600 / 1500: loss 4.649855
iteration 700 / 1500: loss 4.588845
iteration 800 / 1500: loss 4.544648
iteration 900 / 1500: loss 4.431627
iteration 1000 / 1500: loss 4.808049
iteration 1100 / 1500: loss 4.762443
iteration 1200 / 1500: loss 4.910042
iteration 1300 / 1500: loss 4.674157
iteration 1400 / 1500: loss 4.413153
Val_accuracy: 0.124, Current best_val: 0.4
count 540 / 600
iteration 0 / 20: loss 6.212041
iteration 0 / 1500: loss 6.328225
iteration 100 / 1500: loss 5.318247
iteration 200 / 1500: loss 4.828154
iteration 300 / 1500: loss 4.875106
iteration 400 / 1500: loss 4.392539
iteration 500 / 1500: loss 3.933077
iteration 600 / 1500: loss 4.275097
iteration 700 / 1500: loss 3.869797
iteration 800 / 1500: loss 3.559802
iteration 900 / 1500: loss 3.703525
iteration 1000 / 1500: loss 3.576862
iteration 1100 / 1500: loss 3.522995
iteration 1200 / 1500: loss 3.206739
iteration 1300 / 1500: loss 3.284727
iteration 1400 / 1500: loss 3.001821
Val_accuracy: 0.222, Current best_val: 0.4
count 541 / 600
iteration 0 / 20: loss 5.569935
iteration 0 / 1500: loss 6.496516
iteration 100 / 1500: loss 5.218820
iteration 200 / 1500: loss 4.543308
iteration 300 / 1500: loss 4.424047
iteration 400 / 1500: loss 4.301006
iteration 500 / 1500: loss 3.911703
iteration 600 / 1500: loss 3.689644
iteration 700 / 1500: loss 3.666664
iteration 800 / 1500: loss 3.544935
iteration 900 / 1500: loss 3.442770
iteration 1000 / 1500: loss 3.658516
iteration 1100 / 1500: loss 3.191301
iteration 1200 / 1500: loss 3.281390
iteration 1300 / 1500: loss 3.311864
iteration 1400 / 1500: loss 3.123688
Val_accuracy: 0.177, Current best_val: 0.4
count 542 / 600
iteration 0 / 20: loss 19.849377
count 543 / 600
iteration 0 / 20: loss 93.976129
count 544 / 600
iteration 0 / 20: loss 88.511379
count 545 / 600
iteration 0 / 20: loss 23.869959
count 546 / 600
iteration 0 / 20: loss 16.972901
count 547 / 600
iteration 0 / 20: loss 5.255179
iteration 0 / 1500: loss 4.650604
iteration 100 / 1500: loss 2.576719
iteration 200 / 1500: loss 2.486497
iteration 300 / 1500: loss 2.243501
iteration 400 / 1500: loss 2.079000
iteration 500 / 1500: loss 2.013453
iteration 600 / 1500: loss 2.064883
iteration 700 / 1500: loss 1.878705
iteration 800 / 1500: loss 2.076708
iteration 900 / 1500: loss 1.844084
iteration 1000 / 1500: loss 1.720124
iteration 1100 / 1500: loss 1.846510
iteration 1200 / 1500: loss 1.928483
iteration 1300 / 1500: loss 1.903279
iteration 1400 / 1500: loss 1.925779
Val_accuracy: 0.359, Current best_val: 0.4
count 548 / 600
iteration 0 / 20: loss 34.682616
count 549 / 600
iteration 0 / 20: loss 7.158474
iteration 0 / 1500: loss 5.614483
iteration 100 / 1500: loss 5.864991
iteration 200 / 1500: loss 6.415374
iteration 300 / 1500: loss 5.807898
iteration 400 / 1500: loss 6.002941
iteration 500 / 1500: loss 5.712057
iteration 600 / 1500: loss 5.904153
iteration 700 / 1500: loss 5.262043
iteration 800 / 1500: loss 5.741524
iteration 900 / 1500: loss 5.455867
iteration 1000 / 1500: loss 5.652658
iteration 1100 / 1500: loss 5.568050
iteration 1200 / 1500: loss 5.449498
iteration 1300 / 1500: loss 5.672145
iteration 1400 / 1500: loss 5.397865
Val_accuracy: 0.104, Current best_val: 0.4
count 550 / 600
iteration 0 / 20: loss 7.025422
iteration 0 / 1500: loss 7.343236
iteration 100 / 1500: loss 6.897661
iteration 200 / 1500: loss 6.620264
iteration 300 / 1500: loss 5.879894
iteration 400 / 1500: loss 5.535204
iteration 500 / 1500: loss 5.458550
iteration 600 / 1500: loss 5.493696
iteration 700 / 1500: loss 5.210641
iteration 800 / 1500: loss 5.259536
iteration 900 / 1500: loss 5.070388
iteration 1000 / 1500: loss 5.091190
iteration 1100 / 1500: loss 4.914583
iteration 1200 / 1500: loss 4.755593
iteration 1300 / 1500: loss 4.989473
iteration 1400 / 1500: loss 4.861882
Val_accuracy: 0.153, Current best_val: 0.4
count 551 / 600
iteration 0 / 20: loss 10.235583
iteration 0 / 1500: loss 9.495948
iteration 100 / 1500: loss 9.261002
iteration 200 / 1500: loss 9.135343
iteration 300 / 1500: loss 9.488113
iteration 400 / 1500: loss 9.401649
iteration 500 / 1500: loss 9.375656
iteration 600 / 1500: loss 9.188953
iteration 700 / 1500: loss 9.167366
iteration 800 / 1500: loss 9.095836
iteration 900 / 1500: loss 9.016572
iteration 1000 / 1500: loss 9.388723
iteration 1100 / 1500: loss 9.221835
iteration 1200 / 1500: loss 8.748060
iteration 1300 / 1500: loss 9.083260
iteration 1400 / 1500: loss 8.384497
```

```
Val_accuracy: 0.099, Current best_val: 0.4
count 552 / 600
iteration 0 / 20: loss 9.159608
iteration 0 / 1500: loss 9.473573
iteration 100 / 1500: loss 5.016876
iteration 200 / 1500: loss 3.951006
iteration 300 / 1500: loss 3.477614
iteration 400 / 1500: loss 3.072833
iteration 500 / 1500: loss 2.622316
iteration 600 / 1500: loss 2.468425
iteration 700 / 1500: loss 2.341472
iteration 800 / 1500: loss 2.253149
iteration 900 / 1500: loss 2.081898
iteration 1000 / 1500: loss 2.095713
iteration 1100 / 1500: loss 2.035967
iteration 1200 / 1500: loss 2.013567
iteration 1300 / 1500: loss 2.160044
iteration 1400 / 1500: loss 1.900257
Val_accuracy: 0.373, Current best_val: 0.4
count 553 / 600
iteration 0 / 20: loss 470.974901
count 554 / 600
iteration 0 / 20: loss 22.167592
count 555 / 600
iteration 0 / 20: loss 5.844756
iteration 0 / 1500: loss 5.515890
iteration 100 / 1500: loss 3.725322
iteration 200 / 1500: loss 3.796841
iteration 300 / 1500: loss 3.324787
iteration 400 / 1500: loss 3.260668
iteration 500 / 1500: loss 3.160421
iteration 600 / 1500: loss 3.353601
iteration 700 / 1500: loss 2.975248
iteration 800 / 1500: loss 3.006835
iteration 900 / 1500: loss 2.867909
iteration 1000 / 1500: loss 2.574338
iteration 1100 / 1500: loss 2.806780
iteration 1200 / 1500: loss 2.992460
iteration 1300 / 1500: loss 2.762607
iteration 1400 / 1500: loss 2.653039
Val_accuracy: 0.269, Current best_val: 0.4
count 556 / 600
iteration 0 / 20: loss 6.258139
iteration 0 / 1500: loss 5.041362
iteration 100 / 1500: loss 5.394100
iteration 200 / 1500: loss 5.179785
iteration 300 / 1500: loss 5.698507
iteration 400 / 1500: loss 5.139936
iteration 500 / 1500: loss 5.399438
iteration 600 / 1500: loss 5.037434
iteration 700 / 1500: loss 5.464841
iteration 800 / 1500: loss 4.476473
iteration 900 / 1500: loss 4.505966
iteration 1000 / 1500: loss 4.913045
iteration 1100 / 1500: loss 4.793858
iteration 1200 / 1500: loss 4.676127
iteration 1300 / 1500: loss 4.475375
iteration 1400 / 1500: loss 4.098244
Val_accuracy: 0.132, Current best_val: 0.4
count 557 / 600
iteration 0 / 20: loss 25.761830
count 558 / 600
iteration 0 / 20: loss 109.382150
count 559 / 600
iteration 0 / 20: loss 5.324464
iteration 0 / 1500: loss 5.877451
iteration 100 / 1500: loss 5.085727
iteration 200 / 1500: loss 4.598708
iteration 300 / 1500: loss 4.157470
iteration 400 / 1500: loss 4.178037
iteration 500 / 1500: loss 4.018049
iteration 600 / 1500: loss 3.961846
iteration 700 / 1500: loss 3.828156
iteration 800 / 1500: loss 3.921875
iteration 900 / 1500: loss 3.597562
iteration 1000 / 1500: loss 4.004190
iteration 1100 / 1500: loss 3.801292
iteration 1200 / 1500: loss 3.503166
iteration 1300 / 1500: loss 3.694981
iteration 1400 / 1500: loss 3.500508
Val_accuracy: 0.17, Current best_val: 0.4
count 560 / 600
iteration 0 / 20: loss 5.174941
iteration 0 / 1500: loss 5.390654
iteration 100 / 1500: loss 4.485856
iteration 200 / 1500: loss 4.382228
iteration 300 / 1500: loss 4.357032
iteration 400 / 1500: loss 3.554986
iteration 500 / 1500: loss 3.681464
iteration 600 / 1500: loss 3.581804
iteration 700 / 1500: loss 3.244805
iteration 800 / 1500: loss 3.123807
iteration 900 / 1500: loss 3.090391
iteration 1000 / 1500: loss 3.346679
iteration 1100 / 1500: loss 3.224152
iteration 1200 / 1500: loss 2.946432
iteration 1300 / 1500: loss 3.095668
iteration 1400 / 1500: loss 2.843666
Val_accuracy: 0.224, Current best_val: 0.4
count 561 / 600
iteration 0 / 20: loss 9.716438
iteration 0 / 1500: loss 8.794527
iteration 100 / 1500: loss 8.289508
iteration 200 / 1500: loss 7.696777
iteration 300 / 1500: loss 7.577626
iteration 400 / 1500: loss 7.104958
iteration 500 / 1500: loss 7.192136
iteration 600 / 1500: loss 6.737905
iteration 700 / 1500: loss 7.055925
iteration 800 / 1500: loss 7.169756
iteration 900 / 1500: loss 6.702361
iteration 1000 / 1500: loss 6.670703
iteration 1100 / 1500: loss 6.680902
iteration 1200 / 1500: loss 6.669461
iteration 1300 / 1500: loss 6.478923
iteration 1400 / 1500: loss 6.664693
Val_accuracy: 0.204, Current best_val: 0.4
count 562 / 600
iteration 0 / 20: loss 524.706488
count 563 / 600
iteration 0 / 20: loss 55.718308
count 564 / 600
iteration 0 / 20: loss 217.324019
count 565 / 600
iteration 0 / 20: loss 5.753268
iteration 0 / 1500: loss 6.651947
iteration 100 / 1500: loss 2.380883
iteration 200 / 1500: loss 2.112898
iteration 300 / 1500: loss 2.028533
iteration 400 / 1500: loss 1.915474
iteration 500 / 1500: loss 2.114816
iteration 600 / 1500: loss 1.951171
iteration 700 / 1500: loss 1.703465
iteration 800 / 1500: loss 1.876973
iteration 900 / 1500: loss 1.843150
iteration 1000 / 1500: loss 2.151691
iteration 1100 / 1500: loss 1.794840
iteration 1200 / 1500: loss 1.714699
iteration 1300 / 1500: loss 1.708157
iteration 1400 / 1500: loss 1.809081
Val_accuracy: 0.371, Current best_val: 0.4
count 566 / 600
iteration 0 / 20: loss 1215.522626
count 567 / 600
```

```
iteration 0 / 20: loss 6.518083
iteration 0 / 1500: loss 6.959933
iteration 100 / 1500: loss 6.855509
iteration 200 / 1500: loss 6.559180
iteration 300 / 1500: loss 6.212073
iteration 400 / 1500: loss 5.842779
iteration 500 / 1500: loss 5.487511
iteration 600 / 1500: loss 5.392176
iteration 700 / 1500: loss 5.120798
iteration 800 / 1500: loss 5.329340
iteration 900 / 1500: loss 5.040507
iteration 1000 / 1500: loss 5.135500
iteration 1100 / 1500: loss 4.601736
iteration 1200 / 1500: loss 4.970612
iteration 1300 / 1500: loss 5.167351
iteration 1400 / 1500: loss 4.735983
Val_accuracy: 0.181, Current best_val: 0.4
count 568 / 600
iteration 0 / 20: loss 11.076959
iteration 0 / 1500: loss 11.087304
iteration 100 / 1500: loss 10.889897
iteration 200 / 1500: loss 10.983415
iteration 300 / 1500: loss 10.718187
iteration 400 / 1500: loss 10.997034
iteration 500 / 1500: loss 10.508124
iteration 600 / 1500: loss 10.698877
iteration 700 / 1500: loss 10.217950
iteration 800 / 1500: loss 10.867371
iteration 900 / 1500: loss 10.719631
iteration 1000 / 1500: loss 10.804667
iteration 1100 / 1500: loss 10.521384
iteration 1200 / 1500: loss 10.556970
iteration 1300 / 1500: loss 10.512965
iteration 1400 / 1500: loss 10.160148
Val_accuracy: 0.087, Current best_val: 0.4
count 569 / 600
iteration 0 / 20: loss 5.785454
iteration 0 / 1500: loss 5.630705
iteration 100 / 1500: loss 4.538309
iteration 200 / 1500: loss 4.300852
iteration 300 / 1500: loss 3.962645
iteration 400 / 1500: loss 3.883741
iteration 500 / 1500: loss 3.968646
iteration 600 / 1500: loss 3.679204
iteration 700 / 1500: loss 3.663442
iteration 800 / 1500: loss 3.207573
iteration 900 / 1500: loss 3.405129
iteration 1000 / 1500: loss 3.093844
iteration 1100 / 1500: loss 3.275158
iteration 1200 / 1500: loss 3.042370
iteration 1300 / 1500: loss 3.478241
iteration 1400 / 1500: loss 3.192736
Val_accuracy: 0.205, Current best_val: 0.4
count 570 / 600
iteration 0 / 20: loss 5.899397
iteration 0 / 1500: loss 5.830419
iteration 100 / 1500: loss 2.649613
iteration 200 / 1500: loss 2.652498
iteration 300 / 1500: loss 2.434114
iteration 400 / 1500: loss 2.447112
iteration 500 / 1500: loss 2.288973
iteration 600 / 1500: loss 2.420743
iteration 700 / 1500: loss 2.242217
iteration 800 / 1500: loss 2.414697
iteration 900 / 1500: loss 2.281838
iteration 1000 / 1500: loss 2.243049
iteration 1100 / 1500: loss 2.245998
iteration 1200 / 1500: loss 2.123527
iteration 1300 / 1500: loss 2.122287
iteration 1400 / 1500: loss 2.126968
Val_accuracy: 0.377, Current best_val: 0.4
count 571 / 600
iteration 0 / 20: loss 8.880456
iteration 0 / 1500: loss 7.506233
iteration 100 / 1500: loss 7.749140
iteration 200 / 1500: loss 7.575524
iteration 300 / 1500: loss 7.261944
iteration 400 / 1500: loss 7.630306
iteration 500 / 1500: loss 7.444979
iteration 600 / 1500: loss 7.440053
iteration 700 / 1500: loss 7.094946
iteration 800 / 1500: loss 7.660726
iteration 900 / 1500: loss 6.925185
iteration 1000 / 1500: loss 7.149011
iteration 1100 / 1500: loss 6.974121
iteration 1200 / 1500: loss 7.329051
iteration 1300 / 1500: loss 7.015820
iteration 1400 / 1500: loss 7.136569
Val_accuracy: 0.152, Current best_val: 0.4
count 572 / 600
iteration 0 / 20: loss 11.989284
iteration 0 / 1500: loss 11.674270
iteration 100 / 1500: loss 6.970812
iteration 200 / 1500: loss 6.028484
iteration 300 / 1500: loss 5.184280
iteration 400 / 1500: loss 4.751780
iteration 500 / 1500: loss 4.118348
iteration 600 / 1500: loss 3.805129
iteration 700 / 1500: loss 3.358355
iteration 800 / 1500: loss 3.143084
iteration 900 / 1500: loss 2.952045
iteration 1000 / 1500: loss 2.671886
iteration 1100 / 1500: loss 2.606928
iteration 1200 / 1500: loss 2.409309
iteration 1300 / 1500: loss 2.314543
iteration 1400 / 1500: loss 2.233588
Val_accuracy: 0.395, Current best_val: 0.4
count 573 / 600
iteration 0 / 20: loss 6.959253
iteration 0 / 1500: loss 6.119583
iteration 100 / 1500: loss 5.543293
iteration 200 / 1500: loss 4.975782
iteration 300 / 1500: loss 4.284938
iteration 400 / 1500: loss 4.555762
iteration 500 / 1500: loss 4.142081
iteration 600 / 1500: loss 4.288758
iteration 700 / 1500: loss 4.281715
iteration 800 / 1500: loss 3.877953
iteration 900 / 1500: loss 4.326149
iteration 1000 / 1500: loss 3.888804
iteration 1100 / 1500: loss 4.089405
iteration 1200 / 1500: loss 3.663799
iteration 1300 / 1500: loss 3.781343
iteration 1400 / 1500: loss 3.609604
Val_accuracy: 0.184, Current best_val: 0.4
count 574 / 600
iteration 0 / 20: loss 5.588434
iteration 0 / 1500: loss 6.042499
iteration 100 / 1500: loss 4.957523
iteration 200 / 1500: loss 5.071578
iteration 300 / 1500: loss 4.540724
iteration 400 / 1500: loss 3.982494
iteration 500 / 1500: loss 3.800208
iteration 600 / 1500: loss 3.909143
iteration 700 / 1500: loss 4.047552
iteration 800 / 1500: loss 3.790726
iteration 900 / 1500: loss 4.184076
iteration 1000 / 1500: loss 3.880073
iteration 1100 / 1500: loss 3.819199
iteration 1200 / 1500: loss 3.954515
iteration 1300 / 1500: loss 3.781182
iteration 1400 / 1500: loss 3.387097
Val_accuracy: 0.181, Current best_val: 0.4
count 575 / 600
```

```
iteration 0 / 20: loss 1216.322600
count 576 / 600
iteration 0 / 20: loss 6.585786
iteration 0 / 1500: loss 5.869386
iteration 100 / 1500: loss 5.981002
iteration 200 / 1500: loss 5.471243
iteration 300 / 1500: loss 5.310929
iteration 400 / 1500: loss 5.107707
iteration 500 / 1500: loss 4.744175
iteration 600 / 1500: loss 4.654320
iteration 700 / 1500: loss 4.550217
iteration 800 / 1500: loss 5.116321
iteration 900 / 1500: loss 4.848923
iteration 1000 / 1500: loss 4.714175
iteration 1100 / 1500: loss 4.795898
iteration 1200 / 1500: loss 4.580021
iteration 1300 / 1500: loss 4.651852
iteration 1400 / 1500: loss 4.577410
Val_accuracy: 0.128, Current best_val: 0.4
count 577 / 600
iteration 0 / 20: loss 6.093400
iteration 0 / 1500: loss 6.120793
iteration 100 / 1500: loss 5.417521
iteration 200 / 1500: loss 4.569506
iteration 300 / 1500: loss 4.445075
iteration 400 / 1500: loss 4.151252
iteration 500 / 1500: loss 4.004429
iteration 600 / 1500: loss 4.307512
iteration 700 / 1500: loss 4.154092
iteration 800 / 1500: loss 3.649520
iteration 900 / 1500: loss 3.794364
iteration 1000 / 1500: loss 3.765783
iteration 1100 / 1500: loss 3.266908
iteration 1200 / 1500: loss 3.396494
iteration 1300 / 1500: loss 3.618716
iteration 1400 / 1500: loss 3.553423
Val_accuracy: 0.175, Current best_val: 0.4
count 578 / 600
iteration 0 / 20: loss 762.061409
count 579 / 600
iteration 0 / 20: loss 30.596423
count 580 / 600
iteration 0 / 20: loss 184.066293
count 581 / 600
iteration 0 / 20: loss 29.143062
count 582 / 600
iteration 0 / 20: loss 5.790151
iteration 0 / 1500: loss 6.335566
iteration 100 / 1500: loss 6.031359
iteration 200 / 1500: loss 5.853081
iteration 300 / 1500: loss 6.581440
iteration 400 / 1500: loss 6.352616
iteration 500 / 1500: loss 6.084306
iteration 600 / 1500: loss 5.633366
iteration 700 / 1500: loss 5.795366
iteration 800 / 1500: loss 5.414379
iteration 900 / 1500: loss 5.361930
iteration 1000 / 1500: loss 5.624663
iteration 1100 / 1500: loss 5.882162
iteration 1200 / 1500: loss 5.785188
iteration 1300 / 1500: loss 5.249673
iteration 1400 / 1500: loss 5.677337
Val_accuracy: 0.094, Current best_val: 0.4
count 583 / 600
iteration 0 / 20: loss 10.356603
iteration 0 / 1500: loss 9.790344
iteration 100 / 1500: loss 10.435923
iteration 200 / 1500: loss 9.728550
iteration 300 / 1500: loss 9.896670
iteration 400 / 1500: loss 9.784467
iteration 500 / 1500: loss 9.784317
iteration 600 / 1500: loss 9.938108
iteration 700 / 1500: loss 9.880308
iteration 800 / 1500: loss 9.780332
iteration 900 / 1500: loss 10.261200
iteration 1000 / 1500: loss 9.773522
iteration 1100 / 1500: loss 10.135501
iteration 1200 / 1500: loss 9.618342
iteration 1300 / 1500: loss 9.606881
iteration 1400 / 1500: loss 9.482055
Val_accuracy: 0.112, Current best_val: 0.4
count 584 / 600
iteration 0 / 20: loss 564.036093
count 585 / 600
iteration 0 / 20: loss 5.561316
iteration 0 / 1500: loss 5.541619
iteration 100 / 1500: loss 5.643677
iteration 200 / 1500: loss 5.412219
iteration 300 / 1500: loss 5.137671
iteration 400 / 1500: loss 5.441458
iteration 500 / 1500: loss 5.324769
iteration 600 / 1500: loss 5.810235
iteration 700 / 1500: loss 5.070838
iteration 800 / 1500: loss 4.735861
iteration 900 / 1500: loss 5.392403
iteration 1000 / 1500: loss 4.780738
iteration 1100 / 1500: loss 4.547430
iteration 1200 / 1500: loss 5.505749
iteration 1300 / 1500: loss 5.008369
iteration 1400 / 1500: loss 5.170321
Val_accuracy: 0.114, Current best_val: 0.4
count 586 / 600
iteration 0 / 20: loss 85.981874
count 587 / 600
iteration 0 / 20: loss 5.584074
iteration 0 / 1500: loss 5.801682
iteration 100 / 1500: loss 5.062890
iteration 200 / 1500: loss 4.972121
iteration 300 / 1500: loss 4.291820
iteration 400 / 1500: loss 4.366951
iteration 500 / 1500: loss 4.179993
iteration 600 / 1500: loss 3.696081
iteration 700 / 1500: loss 3.801789
iteration 800 / 1500: loss 3.407267
iteration 900 / 1500: loss 3.438868
iteration 1000 / 1500: loss 3.487366
iteration 1100 / 1500: loss 3.435519
iteration 1200 / 1500: loss 2.934046
iteration 1300 / 1500: loss 3.285654
iteration 1400 / 1500: loss 3.385072
Val_accuracy: 0.202, Current best_val: 0.4
count 588 / 600
iteration 0 / 20: loss 125.043346
count 589 / 600
iteration 0 / 20: loss 5.506140
iteration 0 / 1500: loss 5.591574
iteration 100 / 1500: loss 2.963107
iteration 200 / 1500: loss 2.792035
iteration 300 / 1500: loss 2.415319
iteration 400 / 1500: loss 2.347425
iteration 500 / 1500: loss 2.247187
iteration 600 / 1500: loss 2.024143
iteration 700 / 1500: loss 2.184508
iteration 800 / 1500: loss 2.038279
iteration 900 / 1500: loss 2.225038
iteration 1000 / 1500: loss 2.092511
iteration 1100 / 1500: loss 2.093081
iteration 1200 / 1500: loss 2.126574
iteration 1300 / 1500: loss 2.003734
iteration 1400 / 1500: loss 2.050412
Val_accuracy: 0.313, Current best_val: 0.4
count 590 / 600
iteration 0 / 20: loss 7.878814
iteration 0 / 1500: loss 8.734100
iteration 100 / 1500: loss 8.182741
iteration 200 / 1500: loss 8.085453
iteration 300 / 1500: loss 8.031175
iteration 400 / 1500: loss 8.209567
iteration 500 / 1500: loss 8.027398
iteration 600 / 1500: loss 8.112605
iteration 700 / 1500: loss 7.881511
iteration 800 / 1500: loss 8.282482
iteration 900 / 1500: loss 8.576239
iteration 1000 / 1500: loss 8.163419
iteration 1100 / 1500: loss 7.689305
iteration 1200 / 1500: loss 7.907455
iteration 1300 / 1500: loss 8.282386
iteration 1400 / 1500: loss 7.854210
Val_accuracy: 0.095, Current best_val: 0.4
count 591 / 600
iteration 0 / 20: loss 7.447576
iteration 0 / 1500: loss 7.711461
iteration 100 / 1500: loss 4.445672
iteration 200 / 1500: loss 3.992452
iteration 300 / 1500: loss 3.575882
iteration 400 / 1500: loss 3.512659
iteration 500 / 1500: loss 3.535859
iteration 600 / 1500: loss 3.574279
iteration 700 / 1500: loss 3.628979
iteration 800 / 1500: loss 3.333936
iteration 900 / 1500: loss 3.284154
iteration 1000 / 1500: loss 3.115180
iteration 1100 / 1500: loss 3.402176
iteration 1200 / 1500: loss 3.117281
iteration 1300 / 1500: loss 3.244584
iteration 1400 / 1500: loss 2.995724
Val_accuracy: 0.302, Current best_val: 0.4
count 592 / 600
iteration 0 / 20: loss 20.159371
count 593 / 600
iteration 0 / 20: loss 5.601563
iteration 0 / 1500: loss 5.380415
iteration 100 / 1500: loss 3.640192
iteration 200 / 1500: loss 3.295397
iteration 300 / 1500: loss 3.040990
iteration 400 / 1500: loss 2.776912
iteration 500 / 1500: loss 2.828521
iteration 600 / 1500: loss 2.555147
iteration 700 / 1500: loss 2.909226
iteration 800 / 1500: loss 2.600621
iteration 900 / 1500: loss 2.612052
iteration 1000 / 1500: loss 2.516023
iteration 1100 / 1500: loss 2.411208
iteration 1200 / 1500: loss 2.371400
iteration 1300 / 1500: loss 2.727661
iteration 1400 / 1500: loss 2.516376
Val_accuracy: 0.291, Current best_val: 0.4
count 594 / 600
iteration 0 / 20: loss 587.840505
count 595 / 600
iteration 0 / 20: loss 740.697126
count 596 / 600
iteration 0 / 20: loss 5.524951
iteration 0 / 1500: loss 5.480852
iteration 100 / 1500: loss 5.907149
iteration 200 / 1500: loss 5.567194
iteration 300 / 1500: loss 5.362555
iteration 400 / 1500: loss 5.307869
iteration 500 / 1500: loss 5.327659
iteration 600 / 1500: loss 4.846357
iteration 700 / 1500: loss 4.865854
iteration 800 / 1500: loss 5.061042
iteration 900 / 1500: loss 4.750684
iteration 1000 / 1500: loss 4.399214
iteration 1100 / 1500: loss 4.202655
iteration 1200 / 1500: loss 4.846601
iteration 1300 / 1500: loss 4.377544
iteration 1400 / 1500: loss 4.524276
Val_accuracy: 0.119, Current best_val: 0.4
count 597 / 600
iteration 0 / 20: loss 2736.251900
iteration 0 / 1500: loss 2738.181415
iteration 100 / 1500: loss 11.177869
iteration 200 / 1500: loss 13.296270
iteration 300 / 1500: loss 12.182742
iteration 400 / 1500: loss 12.906296
iteration 500 / 1500: loss 11.596030
iteration 600 / 1500: loss 11.166309
iteration 700 / 1500: loss 13.885871
iteration 800 / 1500: loss 13.966535
iteration 900 / 1500: loss 11.889869
iteration 1000 / 1500: loss 12.613137
iteration 1100 / 1500: loss 13.395610
iteration 1200 / 1500: loss 13.257406
iteration 1300 / 1500: loss 11.720931
iteration 1400 / 1500: loss 11.990423
Val_accuracy: 0.069, Current best_val: 0.4
count 598 / 600
iteration 0 / 20: loss 203.355228
count 599 / 600
```

```
iteration 0 / 20: loss 6.427203
iteration 0 / 1500: loss 5.442285
iteration 100 / 1500: loss 4.448925
iteration 200 / 1500: loss 3.816228
iteration 300 / 1500: loss 3.711884
iteration 400 / 1500: loss 3.747962
iteration 500 / 1500: loss 3.358626
iteration 600 / 1500: loss 3.534686
iteration 700 / 1500: loss 3.353290
iteration 800 / 1500: loss 3.069083
iteration 900 / 1500: loss 2.956321
iteration 1000 / 1500: loss 3.376007
iteration 1100 / 1500: loss 3.020090
iteration 1200 / 1500: loss 3.034249
iteration 1300 / 1500: loss 3.083251
iteration 1400 / 1500: loss 3.174707
Val_accuracy: 0.222, Current best_val: 0.4
lr 1.006271e-09 reg 2.084758e+01 train accuracy: 0.104592 val accuracy: 0.119000
lr 1.122786e-09 reg 7.716379e-01 train accuracy: 0.117143 val accuracy: 0.125000
lr 1.158735e-09 reg 2.632166e-01 train accuracy: 0.097245 val accuracy: 0.103000
lr 1.164090e-09 reg 4.152220e+00 train accuracy: 0.085041 val accuracy: 0.088000
lr 1.166069e-09 reg 1.670692e+02 train accuracy: 0.092735 val accuracy: 0.087000
lr 1.170138e-09 reg 5.182658e-01 train accuracy: 0.108959 val accuracy: 0.104000
lr 1.173474e-09 reg 1.386127e+02 train accuracy: 0.105469 val accuracy: 0.099000
lr 1.191514e-09 reg 7.986042e-01 train accuracy: 0.126204 val accuracy: 0.119000
lr 1.218558e-09 reg 1.488680e-01 train accuracy: 0.085959 val accuracy: 0.101000
lr 1.229925e-09 reg 4.612839e-01 train accuracy: 0.125857 val accuracy: 0.123000
lr 1.243290e-09 reg 1.036189e+01 train accuracy: 0.099878 val accuracy: 0.098000
lr 1.273491e-09 reg 2.961887e+00 train accuracy: 0.107714 val accuracy: 0.100000
lr 1.282660e-09 reg 3.596809e-01 train accuracy: 0.105980 val accuracy: 0.105000
lr 1.302355e-09 reg 3.398558e+00 train accuracy: 0.108694 val accuracy: 0.094000
lr 1.310247e-09 reg 1.155026e-01 train accuracy: 0.086367 val accuracy: 0.098000
lr 1.311021e-09 reg 1.046314e+01 train accuracy: 0.113469 val accuracy: 0.104000
lr 1.324325e-09 reg 8.457628e+01 train accuracy: 0.098429 val accuracy: 0.095000
lr 1.334387e-09 reg 1.243468e+01 train accuracy: 0.098122 val accuracy: 0.083000
lr 1.359106e-09 reg 1.931430e+00 train accuracy: 0.077898 val accuracy: 0.079000
lr 1.359724e-09 reg 1.729441e+02 train accuracy: 0.124000 val accuracy: 0.132000
lr 1.393557e-09 reg 1.567837e+02 train accuracy: 0.120531 val accuracy: 0.119000
lr 1.418584e-09 reg 8.317212e-01 train accuracy: 0.092306 val accuracy: 0.087000
lr 1.438465e-09 reg 1.804453e+00 train accuracy: 0.103061 val accuracy: 0.104000
lr 1.488981e-09 reg 4.696294e+01 train accuracy: 0.091959 val accuracy: 0.091000
lr 1.517397e-09 reg 1.454006e+00 train accuracy: 0.099673 val accuracy: 0.095000
lr 1.536245e-09 reg 2.355882e-01 train accuracy: 0.117388 val accuracy: 0.124000
lr 1.590430e-09 reg 1.153720e-01 train accuracy: 0.096265 val accuracy: 0.083000
lr 1.646576e-09 reg 1.062096e+02 train accuracy: 0.098510 val accuracy: 0.094000
lr 1.648201e-09 reg 3.805262e+00 train accuracy: 0.100571 val accuracy: 0.103000
lr 1.705548e-09 reg 1.043852e+01 train accuracy: 0.125347 val accuracy: 0.123000
lr 1.711414e-09 reg 1.407272e-01 train accuracy: 0.110449 val accuracy: 0.104000
lr 1.777025e-09 reg 9.071023e+00 train accuracy: 0.116878 val accuracy: 0.127000
lr 1.825541e-09 reg 1.438961e+01 train accuracy: 0.122612 val accuracy: 0.113000
lr 1.837448e-09 reg 8.142386e+00 train accuracy: 0.109571 val accuracy: 0.102000
lr 1.839322e-09 reg 1.170723e+02 train accuracy: 0.116735 val accuracy: 0.110000
lr 1.868596e-09 reg 2.223756e+00 train accuracy: 0.113490 val accuracy: 0.111000
lr 1.948828e-09 reg 1.662423e-01 train accuracy: 0.122122 val accuracy: 0.126000
lr 2.023832e-09 reg 2.220902e+02 train accuracy: 0.107878 val accuracy: 0.111000
lr 2.090217e-09 reg 1.618795e+02 train accuracy: 0.118776 val accuracy: 0.112000
lr 2.236907e-09 reg 1.874934e+00 train accuracy: 0.113469 val accuracy: 0.135000
lr 2.278833e-09 reg 3.640857e+00 train accuracy: 0.123776 val accuracy: 0.114000
lr 2.323915e-09 reg 5.958133e+00 train accuracy: 0.112347 val accuracy: 0.136000
lr 2.359442e-09 reg 2.145250e+00 train accuracy: 0.111673 val accuracy: 0.114000
lr 2.736941e-09 reg 5.311917e-01 train accuracy: 0.138673 val accuracy: 0.158000
lr 2.795711e-09 reg 8.337068e-01 train accuracy: 0.126714 val accuracy: 0.119000
lr 2.887961e-09 reg 6.025832e+00 train accuracy: 0.129102 val accuracy: 0.135000
lr 3.041549e-09 reg 2.778148e-01 train accuracy: 0.140122 val accuracy: 0.138000
lr 3.102831e-09 reg 1.128728e+02 train accuracy: 0.106796 val accuracy: 0.105000
lr 3.173665e-09 reg 7.783368e+00 train accuracy: 0.129980 val accuracy: 0.134000
lr 3.247621e-09 reg 8.838393e-01 train accuracy: 0.119429 val accuracy: 0.136000
lr 3.317607e-09 reg 6.203394e+00 train accuracy: 0.140694 val accuracy: 0.131000
lr 3.329800e-09 reg 7.877571e+00 train accuracy: 0.108449 val accuracy: 0.128000
lr 3.515251e-09 reg 1.824517e+00 train accuracy: 0.128776 val accuracy: 0.139000
lr 3.652146e-09 reg 1.939275e+01 train accuracy: 0.149469 val accuracy: 0.154000
lr 3.653891e-09 reg 2.482516e+00 train accuracy: 0.127510 val accuracy: 0.143000
lr 3.702908e-09 reg 1.852078e-01 train accuracy: 0.111163 val accuracy: 0.113000
lr 3.824992e-09 reg 9.712432e+01 train accuracy: 0.135388 val accuracy: 0.152000
lr 3.885491e-09 reg 1.345517e+00 train accuracy: 0.132898 val accuracy: 0.148000
lr 4.026963e-09 reg 1.491111e-01 train accuracy: 0.106102 val accuracy: 0.111000
lr 4.222483e-09 reg 1.392736e-01 train accuracy: 0.121735 val accuracy: 0.108000
lr 4.344704e-09 reg 1.091427e-01 train accuracy: 0.124816 val accuracy: 0.122000
lr 4.659291e-09 reg 1.964276e+00 train accuracy: 0.118959 val accuracy: 0.132000
lr 4.687355e-09 reg 1.285583e+01 train accuracy: 0.134000 val accuracy: 0.145000
lr 5.053675e-09 reg 1.505791e+00 train accuracy: 0.147347 val accuracy: 0.171000
lr 5.054896e-09 reg 1.273043e+01 train accuracy: 0.162429 val accuracy: 0.168000
lr 5.325837e-09 reg 1.209887e+00 train accuracy: 0.138469 val accuracy: 0.129000
lr 5.379839e-09 reg 1.002820e+00 train accuracy: 0.131082 val accuracy: 0.134000
lr 5.389162e-09 reg 1.008518e+01 train accuracy: 0.122878 val accuracy: 0.123000
lr 5.534176e-09 reg 1.523889e-01 train accuracy: 0.113837 val accuracy: 0.116000
lr 6.004759e-09 reg 6.840678e+01 train accuracy: 0.130939 val accuracy: 0.126000
lr 6.109967e-09 reg 1.207521e+02 train accuracy: 0.144551 val accuracy: 0.136000
lr 6.117530e-09 reg 1.493680e+00 train accuracy: 0.148286 val accuracy: 0.159000
lr 6.274340e-09 reg 1.831289e-01 train accuracy: 0.139388 val accuracy: 0.150000
lr 6.459480e-09 reg 1.192569e+01 train accuracy: 0.152776 val accuracy: 0.132000
lr 6.938906e-09 reg 2.672320e+01 train accuracy: 0.137122 val accuracy: 0.145000
lr 6.970981e-09 reg 2.992708e+01 train accuracy: 0.151673 val accuracy: 0.142000
lr 7.205056e-09 reg 1.897283e+02 train accuracy: 0.165245 val accuracy: 0.171000
lr 7.214591e-09 reg 1.513708e+01 train accuracy: 0.138837 val accuracy: 0.128000
lr 7.443510e-09 reg 2.677047e-01 train accuracy: 0.120245 val accuracy: 0.124000
lr 7.477651e-09 reg 1.040352e-01 train accuracy: 0.122510 val accuracy: 0.118000
lr 7.536189e-09 reg 3.522397e+00 train accuracy: 0.153551 val accuracy: 0.130000
lr 7.745421e-09 reg 2.029720e+01 train accuracy: 0.121878 val accuracy: 0.102000
lr 8.162139e-09 reg 5.632840e+01 train accuracy: 0.139571 val accuracy: 0.141000
lr 8.329558e-09 reg 1.635784e-01 train accuracy: 0.125061 val accuracy: 0.115000
lr 8.431621e-09 reg 4.417702e+00 train accuracy: 0.155000 val accuracy: 0.157000
lr 8.693447e-09 reg 4.420532e+00 train accuracy: 0.133837 val accuracy: 0.136000
lr 8.760075e-09 reg 3.034805e-01 train accuracy: 0.166122 val accuracy: 0.180000
lr 8.989010e-09 reg 5.012504e+01 train accuracy: 0.148898 val accuracy: 0.152000
lr 9.154429e-09 reg 7.261621e+00 train accuracy: 0.169653 val accuracy: 0.179000
lr 1.010367e-08 reg 3.232446e+01 train accuracy: 0.156204 val accuracy: 0.181000
lr 1.024577e-08 reg 4.344904e+01 train accuracy: 0.157000 val accuracy: 0.171000
lr 1.066678e-08 reg 1.368513e-01 train accuracy: 0.163388 val accuracy: 0.175000
lr 1.078154e-08 reg 1.655673e+01 train accuracy: 0.158102 val accuracy: 0.167000
lr 1.108827e-08 reg 2.461166e-01 train accuracy: 0.162061 val accuracy: 0.165000
lr 1.110825e-08 reg 1.745801e+00 train accuracy: 0.149551 val accuracy: 0.155000
lr 1.174610e-08 reg 1.755030e-01 train accuracy: 0.172694 val accuracy: 0.188000
lr 1.175575e-08 reg 1.404338e+02 train accuracy: 0.146204 val accuracy: 0.164000
lr 1.176218e-08 reg 5.966479e+00 train accuracy: 0.153265 val accuracy: 0.129000
lr 1.235444e-08 reg 1.729712e+01 train accuracy: 0.167306 val accuracy: 0.178000
lr 1.244758e-08 reg 5.231374e+00 train accuracy: 0.158204 val accuracy: 0.144000
lr 1.329733e-08 reg 1.330557e+00 train accuracy: 0.169286 val accuracy: 0.152000
lr 1.404685e-08 reg 1.383109e+00 train accuracy: 0.150388 val accuracy: 0.150000
lr 1.422846e-08 reg 2.424548e+01 train accuracy: 0.144020 val accuracy: 0.140000
lr 1.469332e-08 reg 7.232774e-01 train accuracy: 0.164041 val accuracy: 0.153000
lr 1.547211e-08 reg 3.487562e+01 train accuracy: 0.156102 val accuracy: 0.153000
lr 1.609489e-08 reg 6.529021e+01 train accuracy: 0.169796 val accuracy: 0.185000
lr 1.621610e-08 reg 1.177391e+01 train accuracy: 0.167816 val accuracy: 0.184000
lr 1.629487e-08 reg 1.476662e+00 train accuracy: 0.165571 val accuracy: 0.167000
lr 1.658670e-08 reg 5.324007e-01 train accuracy: 0.171755 val accuracy: 0.169000
lr 1.675186e-08 reg 1.312372e+02 train accuracy: 0.191020 val accuracy: 0.187000
lr 1.685653e-08 reg 7.682543e+00 train accuracy: 0.178653 val accuracy: 0.180000
lr 1.696149e-08 reg 1.709866e+00 train accuracy: 0.191755 val accuracy: 0.192000
lr 1.770991e-08 reg 4.097567e+01 train accuracy: 0.177571 val accuracy: 0.174000
lr 1.771443e-08 reg 7.571419e-01 train accuracy: 0.177735 val accuracy: 0.199000
lr 1.784697e-08 reg 1.250399e+02 train accuracy: 0.158122 val accuracy: 0.176000
lr 1.796486e-08 reg 3.384145e+00 train accuracy: 0.172857 val accuracy: 0.172000
lr 1.816203e-08 reg 2.320027e+01 train accuracy: 0.162388 val accuracy: 0.160000
lr 1.821855e-08 reg 2.077178e+01 train accuracy: 0.189612 val accuracy: 0.204000
lr 1.833793e-08 reg 4.876569e+01 train accuracy: 0.181286 val accuracy: 0.202000
lr 1.836302e-08 reg 2.275845e+00 train accuracy: 0.175000 val accuracy: 0.160000
lr 1.851401e-08 reg 1.703428e-01 train accuracy: 0.173122 val accuracy: 0.191000
lr 1.924939e-08 reg 6.642306e+01 train accuracy: 0.171898 val accuracy: 0.151000
lr 1.986512e-08 reg 5.020277e+00 train accuracy: 0.178898 val accuracy: 0.166000
lr 2.018254e-08 reg 2.765422e-01 train accuracy: 0.160673 val accuracy: 0.170000
lr 2.049527e-08 reg 1.483302e+01 train accuracy: 0.189388 val accuracy: 0.184000
lr 2.121093e-08 reg 1.095196e+00 train accuracy: 0.174980 val accuracy: 0.181000
lr 2.145673e-08 reg 1.437656e-01 train accuracy: 0.191041 val accuracy: 0.191000
lr 2.313815e-08 reg 4.853173e-01 train accuracy: 0.171469 val accuracy: 0.175000
lr 2.341765e-08 reg 9.822364e+00 train accuracy: 0.190837 val accuracy: 0.187000
lr 2.342443e-08 reg 2.483003e+00 train accuracy: 0.188143 val accuracy: 0.189000
lr 2.372839e-08 reg 1.730538e-01 train accuracy: 0.186122 val accuracy: 0.179000
lr 2.428436e-08 reg 4.376317e+00 train accuracy: 0.191939 val accuracy: 0.203000
lr 2.431956e-08 reg 3.768817e+00 train accuracy: 0.173388 val accuracy: 0.175000
lr 2.469678e-08 reg 3.468446e-01 train accuracy: 0.191959 val accuracy: 0.184000
lr 2.542102e-08 reg 1.068302e+02 train accuracy: 0.180694 val accuracy: 0.204000
lr 2.576709e-08 reg 1.113373e+00 train accuracy: 0.190163 val accuracy: 0.190000
lr 2.725295e-08 reg 1.885953e+01 train accuracy: 0.193184 val accuracy: 0.178000
lr 2.776719e-08 reg 6.473389e-01 train accuracy: 0.198122 val accuracy: 0.222000
lr 2.950791e-08 reg 8.629665e+00 train accuracy: 0.207061 val accuracy: 0.213000
lr 3.148473e-08 reg 3.653823e-01 train accuracy: 0.195000 val accuracy: 0.205000
lr 3.149607e-08 reg 3.217615e+00 train accuracy: 0.187837 val accuracy: 0.177000
lr 3.217426e-08 reg 2.207594e+00 train accuracy: 0.216408 val accuracy: 0.222000
lr 3.326376e-08 reg 2.333450e+01 train accuracy: 0.189327 val accuracy: 0.193000
lr 3.359355e-08 reg 1.517784e+02 train accuracy: 0.206020 val accuracy: 0.198000
lr 3.388587e-08 reg 7.364067e-01 train accuracy: 0.198816 val accuracy: 0.178000
lr 3.478218e-08 reg 2.162022e+02 train accuracy: 0.194347 val accuracy: 0.206000
lr 3.637704e-08 reg 2.575167e-01 train accuracy: 0.203571 val accuracy: 0.191000
lr 3.664618e-08 reg 6.394610e+01 train accuracy: 0.189469 val accuracy: 0.185000
lr 3.817198e-08 reg 3.048095e+00 train accuracy: 0.202490 val accuracy: 0.211000
lr 4.017365e-08 reg 3.171790e+01 train accuracy: 0.207816 val accuracy: 0.224000
lr 4.168013e-08 reg 4.871376e-01 train accuracy: 0.202796 val accuracy: 0.224000
lr 4.215460e-08 reg 5.750414e+01 train accuracy: 0.202041 val accuracy: 0.200000
lr 4.287973e-08 reg 3.971816e+00 train accuracy: 0.201408 val accuracy: 0.213000
lr 4.634317e-08 reg 6.821264e+00 train accuracy: 0.203408 val accuracy: 0.202000
lr 4.850175e-08 reg 2.203930e+02 train accuracy: 0.216531 val accuracy: 0.230000
lr 5.047111e-08 reg 1.218025e+00 train accuracy: 0.221796 val accuracy: 0.222000
lr 5.524526e-08 reg 6.386852e-01 train accuracy: 0.221429 val accuracy: 0.239000
lr 5.585240e-08 reg 1.282884e+00 train accuracy: 0.217469 val accuracy: 0.218000
lr 5.731248e-08 reg 7.166381e+00 train accuracy: 0.216367 val accuracy: 0.236000
lr 5.844408e-08 reg 2.129365e+02 train accuracy: 0.223735 val accuracy: 0.216000
lr 5.853375e-08 reg 3.811386e-01 train accuracy: 0.223796 val accuracy: 0.234000
lr 6.069388e-08 reg 1.740607e-01 train accuracy: 0.227184 val accuracy: 0.218000
lr 6.180095e-08 reg 1.527586e+01 train accuracy: 0.229102 val accuracy: 0.231000
lr 6.234448e-08 reg 1.073899e+01 train accuracy: 0.234000 val accuracy: 0.223000
lr 6.256952e-08 reg 1.426339e+01 train accuracy: 0.229735 val accuracy: 0.242000
lr 6.611197e-08 reg 1.949790e+00 train accuracy: 0.230612 val accuracy: 0.223000
lr 6.957373e-08 reg 8.030241e-01 train accuracy: 0.232469 val accuracy: 0.227000
lr 7.040975e-08 reg 2.229105e+01 train accuracy: 0.234163 val accuracy: 0.238000
lr 7.138558e-08 reg 1.547243e+02 train accuracy: 0.235735 val accuracy: 0.259000
lr 7.579171e-08 reg 4.504185e+01 train accuracy: 0.240898 val accuracy: 0.252000
lr 7.962929e-08 reg 1.870019e+01 train accuracy: 0.236837 val accuracy: 0.237000
lr 8.136343e-08 reg 1.428187e+01 train accuracy: 0.225959 val accuracy: 0.219000
lr 8.231762e-08 reg 1.612221e+02 train accuracy: 0.235755 val accuracy: 0.234000
lr 8.254427e-08 reg 3.815365e-01 train accuracy: 0.242735 val accuracy: 0.256000
lr 8.508929e-08 reg 2.210789e+02 train accuracy: 0.251327 val accuracy: 0.256000
lr 8.695813e-08 reg 3.812681e+00 train accuracy: 0.242592 val accuracy: 0.235000
lr 8.940242e-08 reg 3.605952e+01 train accuracy: 0.250592 val accuracy: 0.268000
lr 9.016312e-08 reg 2.915779e-01 train accuracy: 0.237531 val accuracy: 0.246000
lr 9.023229e-08 reg 1.386859e+01 train accuracy: 0.246184 val accuracy: 0.271000
lr 9.275134e-08 reg 6.869551e+00 train accuracy: 0.242102 val accuracy: 0.256000
lr 9.299130e-08 reg 2.649068e-01 train accuracy: 0.236163 val accuracy: 0.233000
lr 9.412366e-08 reg 5.749376e+00 train accuracy: 0.246939 val accuracy: 0.239000
lr 9.425438e-08 reg 9.370094e+00 train accuracy: 0.246429 val accuracy: 0.258000
lr 9.538179e-08 reg 9.651517e-01 train accuracy: 0.241592 val accuracy: 0.260000
lr 9.842234e-08 reg 5.320544e-01 train accuracy: 0.244980 val accuracy: 0.256000
lr 9.937420e-08 reg 8.571533e+01 train accuracy: 0.250449 val accuracy: 0.243000
lr 1.017509e-07 reg 1.775931e+01 train accuracy: 0.245898 val accuracy: 0.255000
lr 1.021918e-07 reg 3.778091e+00 train accuracy: 0.254367 val accuracy: 0.266000
lr 1.100256e-07 reg 2.321110e+01 train accuracy: 0.250776 val accuracy: 0.258000
lr 1.140930e-07 reg 7.996452e+00 train accuracy: 0.256592 val accuracy: 0.269000
lr 1.141255e-07 reg 5.157073e+01 train accuracy: 0.262000 val accuracy: 0.252000
lr 1.142830e-07 reg 6.160518e+01 train accuracy: 0.247408 val accuracy: 0.262000
lr 1.178210e-07 reg 1.826112e-01 train accuracy: 0.251408 val accuracy: 0.256000
lr 1.282909e-07 reg 1.542291e+01 train accuracy: 0.255612 val accuracy: 0.259000
lr 1.314846e-07 reg 1.897648e+01 train accuracy: 0.256898 val accuracy: 0.254000
lr 1.321361e-07 reg 1.558901e+02 train accuracy: 0.263429 val accuracy: 0.264000
lr 1.356602e-07 reg 4.801257e-01 train accuracy: 0.262429 val accuracy: 0.251000
lr 1.413433e-07 reg 5.059353e+00 train accuracy: 0.266306 val accuracy: 0.267000
lr 1.414711e-07 reg 1.677496e+02 train accuracy: 0.261837 val accuracy: 0.262000
lr 1.441952e-07 reg 5.425427e-01 train accuracy: 0.265306 val accuracy: 0.297000
lr 1.468192e-07 reg 4.657467e+01 train accuracy: 0.266061 val accuracy: 0.270000
lr 1.482225e-07 reg 4.215063e+01 train accuracy: 0.269020 val accuracy: 0.278000
lr 1.593168e-07 reg 4.323666e+00 train accuracy: 0.265020 val accuracy: 0.266000
lr 1.629416e-07 reg 1.465772e+00 train accuracy: 0.272265 val accuracy: 0.243000
lr 1.648284e-07 reg 2.520130e-01 train accuracy: 0.269735 val accuracy: 0.274000
lr 1.710209e-07 reg 6.273329e-01 train accuracy: 0.272837 val accuracy: 0.291000
lr 1.757596e-07 reg 6.526670e+00 train accuracy: 0.269612 val accuracy: 0.284000
lr 1.923111e-07 reg 1.268513e+02 train accuracy: 0.276633 val accuracy: 0.281000
lr 1.928598e-07 reg 4.891293e+01 train accuracy: 0.275224 val accuracy: 0.284000
lr 1.933998e-07 reg 1.159060e+00 train accuracy: 0.278000 val accuracy: 0.271000
lr 1.981697e-07 reg 9.935506e+00 train accuracy: 0.278918 val accuracy: 0.286000
lr 2.124844e-07 reg 4.071594e+00 train accuracy: 0.280184 val accuracy: 0.283000
lr 2.193250e-07 reg 1.640305e+02 train accuracy: 0.292041 val accuracy: 0.311000
lr 2.238858e-07 reg 1.551752e+02 train accuracy: 0.292633 val accuracy: 0.298000
lr 2.369237e-07 reg 6.164940e+00 train accuracy: 0.290000 val accuracy: 0.294000
lr 2.416749e-07 reg 3.420499e+00 train accuracy: 0.278776 val accuracy: 0.294000
lr 2.418939e-07 reg 3.608052e+00 train accuracy: 0.288612 val accuracy: 0.278000
lr 2.495616e-07 reg 4.402089e+00 train accuracy: 0.282837 val accuracy: 0.285000
lr 2.517742e-07 reg 3.391884e-01 train accuracy: 0.284878 val accuracy: 0.312000
lr 2.701048e-07 reg 3.327904e+01 train accuracy: 0.288000 val accuracy: 0.279000
lr 2.799190e-07 reg 1.130735e+01 train accuracy: 0.291878 val accuracy: 0.286000
lr 2.859432e-07 reg 1.050488e+01 train accuracy: 0.297653 val accuracy: 0.276000
lr 3.175389e-07 reg 1.155675e-01 train accuracy: 0.301531 val accuracy: 0.280000
lr 3.182445e-07 reg 2.627386e+02 train accuracy: 0.311898 val accuracy: 0.300000
lr 3.202449e-07 reg 3.348787e+01 train accuracy: 0.297490 val accuracy: 0.302000
lr 3.219021e-07 reg 3.528228e-01 train accuracy: 0.297429 val accuracy: 0.328000
lr 3.237717e-07 reg 3.209366e+01 train accuracy: 0.303122 val accuracy: 0.318000
lr 3.288639e-07 reg 1.251875e+00 train accuracy: 0.305184 val accuracy: 0.308000
lr 3.289750e-07 reg 1.074246e+02 train accuracy: 0.304959 val accuracy: 0.313000
lr 3.341978e-07 reg 7.510773e+01 train accuracy: 0.301041 val accuracy: 0.305000
lr 3.349401e-07 reg 1.425807e-01 train accuracy: 0.299571 val accuracy: 0.288000
lr 3.579419e-07 reg 1.134027e+00 train accuracy: 0.300224 val accuracy: 0.295000
lr 3.580343e-07 reg 1.606683e+02 train accuracy: 0.318694 val accuracy: 0.328000
lr 3.582156e-07 reg 1.371188e-01 train accuracy: 0.301898 val accuracy: 0.297000
lr 3.652759e-07 reg 1.427481e+01 train accuracy: 0.301388 val accuracy: 0.304000
lr 3.861567e-07 reg 6.391182e+01 train accuracy: 0.313306 val accuracy: 0.309000
lr 3.993919e-07 reg 4.783002e+00 train accuracy: 0.312449 val accuracy: 0.301000
lr 4.079893e-07 reg 3.372662e+01 train accuracy: 0.318694 val accuracy: 0.321000
lr 4.178310e-07 reg 4.718256e+00 train accuracy: 0.310143 val accuracy: 0.302000
lr 4.207088e-07 reg 7.497059e-01 train accuracy: 0.314122 val accuracy: 0.310000
lr 4.671924e-07 reg 2.758391e+01 train accuracy: 0.318857 val accuracy: 0.311000
lr 4.708254e-07 reg 5.899544e-01 train accuracy: 0.319082 val accuracy: 0.301000
lr 4.730195e-07 reg 5.861657e-01 train accuracy: 0.318286 val accuracy: 0.306000
lr 4.784024e-07 reg 1.074977e+01 train accuracy: 0.312224 val accuracy: 0.322000
lr 4.833299e-07 reg 5.013590e+00 train accuracy: 0.321306 val accuracy: 0.329000
lr 4.916344e-07 reg 1.244227e+02 train accuracy: 0.329449 val accuracy: 0.318000
lr 5.002670e-07 reg 1.053763e+01 train accuracy: 0.321102 val accuracy: 0.307000
lr 5.164464e-07 reg 8.322936e+01 train accuracy: 0.327286 val accuracy: 0.332000
lr 5.380166e-07 reg 2.249190e-01 train accuracy: 0.320755 val accuracy: 0.329000
lr 5.666962e-07 reg 1.197668e+00 train accuracy: 0.322429 val accuracy: 0.296000
lr 5.747422e-07 reg 8.061068e-01 train accuracy: 0.323449 val accuracy: 0.323000
lr 5.760205e-07 reg 7.166099e+01 train accuracy: 0.332265 val accuracy: 0.334000
lr 5.761063e-07 reg 4.461308e-01 train accuracy: 0.321816 val accuracy: 0.315000
lr 5.906125e-07 reg 6.083906e+01 train accuracy: 0.335857 val accuracy: 0.321000
lr 5.941781e-07 reg 2.466701e+01 train accuracy: 0.327286 val accuracy: 0.326000
lr 5.949776e-07 reg 1.792229e-01 train accuracy: 0.327184 val accuracy: 0.335000
lr 6.001807e-07 reg 8.373804e-01 train accuracy: 0.333469 val accuracy: 0.330000
lr 6.023222e-07 reg 4.641505e-01 train accuracy: 0.322490 val accuracy: 0.331000
lr 6.353906e-07 reg 6.586586e+01 train accuracy: 0.336633 val accuracy: 0.326000
lr 6.734034e-07 reg 2.116190e-01 train accuracy: 0.327633 val accuracy: 0.313000
lr 7.939728e-07 reg 1.319337e-01 train accuracy: 0.334306 val accuracy: 0.332000
lr 8.057621e-07 reg 5.050441e+01 train accuracy: 0.348653 val accuracy: 0.351000
lr 8.077242e-07 reg 1.331168e+00 train accuracy: 0.344939 val accuracy: 0.338000
lr 8.285154e-07 reg 4.826155e+00 train accuracy: 0.340163 val accuracy: 0.329000
lr 8.692372e-07 reg 5.101309e-01 train accuracy: 0.338980 val accuracy: 0.312000
lr 8.991291e-07 reg 6.342219e+00 train accuracy: 0.340939 val accuracy: 0.348000
lr 9.108588e-07 reg 2.675295e+02 train accuracy: 0.379388 val accuracy: 0.384000
lr 9.117195e-07 reg 6.541749e-01 train accuracy: 0.343327 val accuracy: 0.323000
lr 9.583895e-07 reg 2.907399e+02 train accuracy: 0.390755 val accuracy: 0.384000
lr 1.029077e-06 reg 6.374565e+00 train accuracy: 0.342551 val accuracy: 0.344000
lr 1.083148e-06 reg 1.504529e+01 train accuracy: 0.355429 val accuracy: 0.327000
lr 1.112290e-06 reg 7.873330e-01 train accuracy: 0.351122 val accuracy: 0.337000
lr 1.149222e-06 reg 2.224590e+02 train accuracy: 0.392224 val accuracy: 0.381000
lr 1.182514e-06 reg 2.608551e+01 train accuracy: 0.359531 val accuracy: 0.354000
lr 1.217104e-06 reg 9.593041e-01 train accuracy: 0.356122 val accuracy: 0.362000
lr 1.248427e-06 reg 1.561656e+01 train accuracy: 0.362469 val accuracy: 0.354000
lr 1.298846e-06 reg 1.677720e+02 train accuracy: 0.391653 val accuracy: 0.382000
lr 1.303006e-06 reg 1.238329e+01 train accuracy: 0.361592 val accuracy: 0.340000
lr 1.332303e-06 reg 1.648740e-01 train accuracy: 0.360571 val accuracy: 0.364000
lr 1.332729e-06 reg 2.512552e+01 train accuracy: 0.364388 val accuracy: 0.357000
lr 1.397285e-06 reg 5.557636e-01 train accuracy: 0.362388 val accuracy: 0.372000
lr 1.400806e-06 reg 6.136010e+00 train accuracy: 0.363469 val accuracy: 0.382000
lr 1.437096e-06 reg 5.362161e-01 train accuracy: 0.360143 val accuracy: 0.327000
lr 1.499085e-06 reg 1.117472e+00 train accuracy: 0.365347 val accuracy: 0.355000
lr 1.499479e-06 reg 2.244208e-01 train accuracy: 0.364959 val accuracy: 0.372000
lr 1.529129e-06 reg 3.051809e-01 train accuracy: 0.364449 val accuracy: 0.366000
lr 1.549605e-06 reg 1.335644e+02 train accuracy: 0.397347 val accuracy: 0.377000
lr 1.608592e-06 reg 3.023669e+00 train accuracy: 0.364714 val accuracy: 0.341000
lr 1.662179e-06 reg 1.040978e-01 train accuracy: 0.368204 val accuracy: 0.373000
lr 1.685772e-06 reg 3.626867e+01 train accuracy: 0.378143 val accuracy: 0.372000
lr 1.742761e-06 reg 1.570414e-01 train accuracy: 0.369490 val accuracy: 0.359000
lr 1.977482e-06 reg 2.238222e-01 train accuracy: 0.373020 val accuracy: 0.349000
lr 2.018224e-06 reg 9.149266e+01 train accuracy: 0.397633 val accuracy: 0.400000
lr 2.097665e-06 reg 1.260443e+01 train accuracy: 0.370286 val accuracy: 0.365000
lr 2.141809e-06 reg 3.234372e+00 train accuracy: 0.382327 val accuracy: 0.361000
lr 2.213529e-06 reg 3.888370e-01 train accuracy: 0.370694 val accuracy: 0.372000
lr 2.243644e-06 reg 1.858631e+02 train accuracy: 0.402510 val accuracy: 0.395000
lr 2.437482e-06 reg 2.894343e-01 train accuracy: 0.379673 val accuracy: 0.374000
lr 2.472442e-06 reg 1.138666e+00 train accuracy: 0.379082 val accuracy: 0.372000
lr 2.487350e-06 reg 2.230821e+04 train accuracy: 0.295755 val accuracy: 0.291000
lr 2.602788e-06 reg 1.593426e+01 train accuracy: 0.385245 val accuracy: 0.374000
lr 2.617513e-06 reg 1.668332e+01 train accuracy: 0.391980 val accuracy: 0.364000
lr 2.793226e-06 reg 8.881007e+01 train accuracy: 0.404122 val accuracy: 0.389000
lr 2.803629e-06 reg 1.131438e-01 train accuracy: 0.380735 val accuracy: 0.367000
lr 2.826398e-06 reg 6.463563e+00 train accuracy: 0.387755 val accuracy: 0.373000
lr 2.871064e-06 reg 1.493906e+01 train accuracy: 0.387592 val accuracy: 0.377000
lr 2.879942e-06 reg 7.998518e-01 train accuracy: 0.387020 val accuracy: 0.368000
lr 2.887693e-06 reg 3.518723e-01 train accuracy: 0.379918 val accuracy: 0.359000
lr 2.913550e-06 reg 9.094852e-01 train accuracy: 0.385286 val accuracy: 0.375000
lr 2.943689e-06 reg 1.184940e-01 train accuracy: 0.379061 val accuracy: 0.345000
lr 2.978352e-06 reg 1.901794e-01 train accuracy: 0.375286 val accuracy: 0.353000
lr 2.984813e-06 reg 1.817002e-01 train accuracy: 0.381796 val accuracy: 0.379000
lr 3.083398e-06 reg 6.817460e-01 train accuracy: 0.391020 val accuracy: 0.367000
lr 3.175674e-06 reg 1.079978e+00 train accuracy: 0.386939 val accuracy: 0.359000
lr 3.234927e-06 reg 4.345804e-01 train accuracy: 0.390939 val accuracy: 0.367000
lr 3.349025e-06 reg 3.018782e+00 train accuracy: 0.382265 val accuracy: 0.364000
lr 3.390594e-06 reg 3.134018e+02 train accuracy: 0.401469 val accuracy: 0.389000
lr 3.573747e-06 reg 6.584156e-01 train accuracy: 0.392816 val accuracy: 0.371000
lr 3.584340e-06 reg 1.802507e+00 train accuracy: 0.391735 val accuracy: 0.378000
lr 3.806591e-06 reg 3.267480e+02 train accuracy: 0.401224 val accuracy: 0.400000
lr 3.983716e-06 reg 2.346155e+04 train accuracy: 0.267388 val accuracy: 0.265000
lr 4.642161e-06 reg 5.064288e-01 train accuracy: 0.398265 val accuracy: 0.374000
lr 4.781585e-06 reg 4.745536e+01 train accuracy: 0.395388 val accuracy: 0.371000
lr 5.093724e-06 reg 1.654616e+02 train accuracy: 0.380306 val accuracy: 0.364000
lr 5.213973e-06 reg 6.147569e-01 train accuracy: 0.380592 val accuracy: 0.362000
lr 5.276446e-06 reg 2.655995e+04 train accuracy: 0.204633 val accuracy: 0.202000
lr 5.474453e-06 reg 1.089403e-01 train accuracy: 0.383653 val accuracy: 0.371000
lr 5.717508e-06 reg 1.349848e+02 train accuracy: 0.371490 val accuracy: 0.373000
lr 6.150878e-06 reg 8.921673e+04 train accuracy: 0.078224 val accuracy: 0.069000
lr 6.167904e-06 reg 1.136816e+02 train accuracy: 0.403020 val accuracy: 0.393000
lr 6.211588e-06 reg 2.830590e+04 train accuracy: 0.199490 val accuracy: 0.197000
lr 6.637594e-06 reg 1.012427e+00 train accuracy: 0.331265 val accuracy: 0.335000
lr 6.929431e-06 reg 1.138278e+02 train accuracy: 0.321939 val accuracy: 0.329000
lr 7.008686e-06 reg 4.105960e+04 train accuracy: 0.170571 val accuracy: 0.170000
lr 7.114093e-06 reg 2.416073e+02 train accuracy: 0.333673 val accuracy: 0.323000
lr 7.337155e-06 reg 3.577041e-01 train accuracy: 0.375265 val accuracy: 0.334000
lr 7.551216e-06 reg 1.450362e+02 train accuracy: 0.294653 val accuracy: 0.303000
lr 7.618074e-06 reg 1.681925e+00 train accuracy: 0.347204 val accuracy: 0.324000
lr 7.646307e-06 reg 5.985291e+00 train accuracy: 0.375653 val accuracy: 0.340000
lr 7.764939e-06 reg 3.133170e+01 train accuracy: 0.352490 val accuracy: 0.333000
lr 7.836802e-06 reg 3.606604e+02 train accuracy: 0.291163 val accuracy: 0.293000
lr 7.843446e-06 reg 4.151423e+02 train accuracy: 0.303306 val accuracy: 0.305000
lr 7.921393e-06 reg 1.327214e-01 train accuracy: 0.383388 val accuracy: 0.363000
lr 8.174736e-06 reg 2.451170e+00 train accuracy: 0.309224 val accuracy: 0.302000
lr 8.724985e-06 reg 4.454271e+04 train accuracy: 0.089776 val accuracy: 0.093000
lr 8.805545e-06 reg 2.790683e+02 train accuracy: 0.343224 val accuracy: 0.327000
lr 8.915927e-06 reg 2.658950e+01 train accuracy: 0.359163 val accuracy: 0.345000
lr 9.085431e-06 reg 1.166034e-01 train accuracy: 0.366041 val accuracy: 0.367000
lr 9.086590e-06 reg 2.454653e+02 train accuracy: 0.262347 val accuracy: 0.258000
lr 9.290835e-06 reg 1.302663e+01 train accuracy: 0.309837 val accuracy: 0.294000
lr 9.682181e-06 reg 6.603098e-01 train accuracy: 0.324959 val accuracy: 0.292000
lr 9.814123e-06 reg 2.218550e-01 train accuracy: 0.345959 val accuracy: 0.346000
best validation accuracy achieved during cross-validation: 0.400000
```

[33]:
```python
# 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_accuracy, ))
```

```
softmax on raw pixels final test set accuracy: 0.379000
```

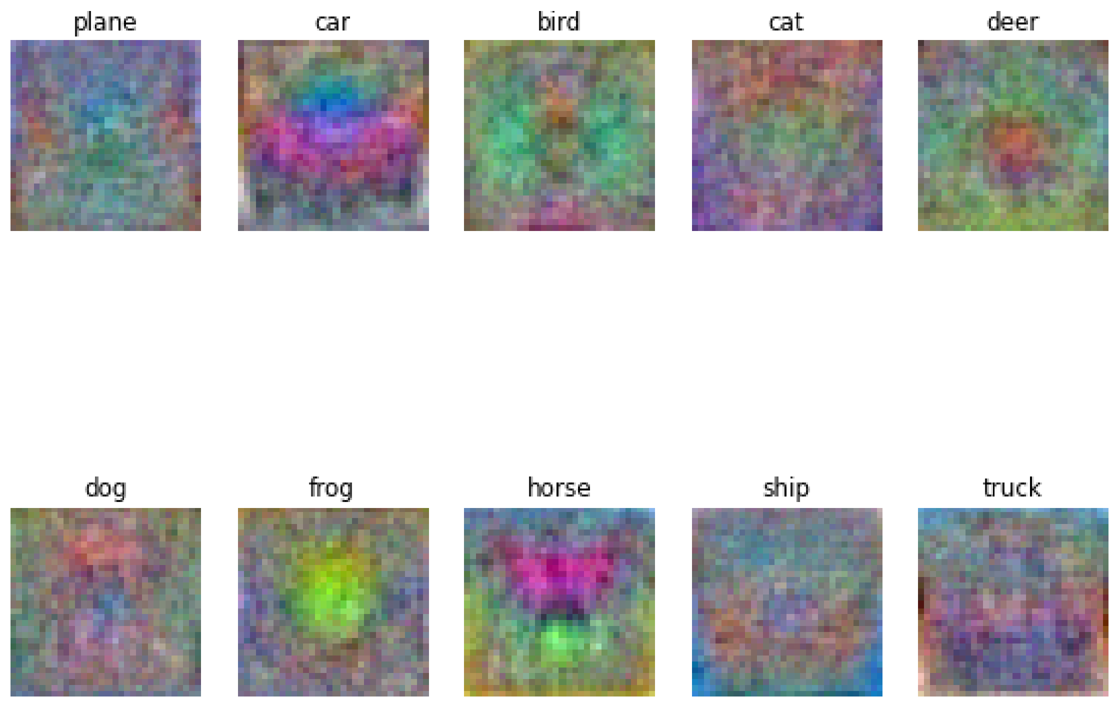
Inline Question 2 - *True or False*

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

Your Answer :

YourExplanation :

[34]:
```python
# 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])
```



two_layer_net

August 6, 2021

[2]:
```
# This mounts your Google Drive to the Colab VM.
from google.colab import drive
drive.mount('/content/drive', force_remount=True)

# Enter the foldername in your Drive where you have saved the unzipped
# assignment folder, e.g. 'cs231n/assignments/assignment1/'
FOLDERNAME = 'CS231N/assignment1'
assert FOLDERNAME is not None, "[!] Enter the foldername."

# Now that we've mounted your Drive, this ensures that
# the Python interpreter of the Colab VM can load
# python files from within it.
import sys
sys.path.append('/content/drive/My Drive/{}'.format(FOLDERNAME))

# This downloads the CIFAR-10 dataset to your Drive
# if it doesn't already exist.
%cd drive/My\ Drive/$FOLDERNAME/cs231n/datasets/
!bash get_datasets.sh
%cd /content/drive/My\ Drive/$FOLDERNAME
```

```
Mounted at /content/drive
/content/drive/My Drive/CS231N/assignment1/cs231n/datasets
/content/drive/My Drive/CS231N/assignment1
```

1 Fully-Connected Neural Nets

In this exercise we will implement fully-connected networks using a modular approach. For each layer we will implement a `forward` and a `backward` function. The `forward` function will receive inputs, weights, and other parameters and will return both an output and a `cache` object storing data needed for the backward pass, like this:

```
def layer_forward(x, w):
  """ Receive inputs x and weights w """
  # Do some computations ...
  z = # ... some intermediate value
  # Do some more computations ...
  out = # the output

  cache = (x, w, z, out) # Values we need to compute gradients

  return out, cache
```

The backward pass will receive upstream derivatives and the cache object, and will return gradients with respect to the inputs and weights, like this:

```
def layer_backward(dout, cache):
  """
  Receive dout (derivative of loss with respect to outputs) and cache,
  and compute derivative with respect to inputs.
  """
  # Unpack cache values
  x, w, z, out = cache

  # Use values in cache to compute derivatives
  dx = # Derivative of loss with respect to x
  dw = # Derivative of loss with respect to w

  return dx, dw
```

After implementing a bunch of layers this way, we will be able to easily combine them to build classifiers with different architectures.

[3]:
```
# As usual, a bit of setup
from __future__ import print_function
import time
import numpy as np
import matplotlib.pyplot as plt
from cs231n.classifiers.fc_net import *
from cs231n.data_utils import get_CIFAR10_data
from cs231n.gradient_check import eval_numerical_gradient,␣
 ↪eval_numerical_gradient_array
from cs231n.solver import Solver

%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 external modules
# see http://stackoverflow.com/questions/1907993/
 ↪autoreload-of-modules-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.abs(y))))
```

[4]:
```
# Load the (preprocessed) CIFAR10 data.

data = get_CIFAR10_data()
for k, v in list(data.items()):
  print(('%s: ' % k, v.shape))
```

```
('X_train: ', (49000, 3, 32, 32))
('y_train: ', (49000,))
('X_val: ', (1000, 3, 32, 32))
('y_val: ', (1000,))
('X_test: ', (1000, 3, 32, 32))
('y_test: ', (1000,))
```

2 Affine layer: forward

Open the file cs231n/layers.py and implement the affine_forward function.

Once you are done you can test your implementaion by running the following:

[5]:
```
# Test the affine_forward function

num_inputs = 2
input_shape = (4, 5, 6)
output_dim = 3

input_size = num_inputs * np.prod(input_shape)
weight_size = output_dim * np.prod(input_shape)

x = np.linspace(-0.1, 0.5, num=input_size).reshape(num_inputs, *input_shape)
w = np.linspace(-0.2, 0.3, num=weight_size).reshape(np.prod(input_shape),␣
 ↪output_dim)
b = np.linspace(-0.3, 0.1, num=output_dim)

out, _ = affine_forward(x, w, b)
correct_out = np.array([[ 1.49834967,  1.70660132,  1.91485297],
                        [ 3.25553199,  3.5141327,   3.77273342]])

# Compare your output with ours. The error should be around e-9 or less.
print('Testing affine_forward function:')
print('difference: ', rel_error(out, correct_out))
```

```
Testing affine_forward function:
difference:  9.769849468192957e-10
```

3 Affine layer: backward

Now implement the `affine_backward` function and test your implementation using numeric gradient checking.

[6]:
```
# Test the affine_backward function
np.random.seed(231)
x = np.random.randn(10, 2, 3)
w = np.random.randn(6, 5)
b = np.random.randn(5)
dout = np.random.randn(10, 5)

dx_num = eval_numerical_gradient_array(lambda x: affine_forward(x, w, b)[0], x,␣
 ↪dout)
dw_num = eval_numerical_gradient_array(lambda w: affine_forward(x, w, b)[0], w,␣
 ↪dout)
db_num = eval_numerical_gradient_array(lambda b: affine_forward(x, w, b)[0], b,␣
 ↪dout)

_, cache = affine_forward(x, w, b)
dx, dw, db = affine_backward(dout, cache)

# The error should be around e-10 or less
print('Testing affine_backward function:')
print('dx error: ', rel_error(dx_num, dx))
print('dw error: ', rel_error(dw_num, dw))
print('db error: ', rel_error(db_num, db))
```

```
Testing affine_backward function:
dx error:  5.399100368651805e-11
dw error:  9.904211865398145e-11
db error:  2.4122867568119087e-11
```

4 ReLU activation: forward

Implement the forward pass for the ReLU activation function in the `relu_forward` function and test your implementation using the following:

[7]:
```
# Test the relu_forward function

x = np.linspace(-0.5, 0.5, num=12).reshape(3, 4)

out, _ = relu_forward(x)
correct_out = np.array([[ 0.,          0.,          0.,          0.,         ],
                        [ 0.,          0.,          0.04545455,  0.13636364,],
                        [ 0.22727273,  0.31818182,  0.40909091,  0.5,       ]])

# Compare your output with ours. The error should be on the order of e-8
```

```
print('Testing relu_forward function:')
print('difference: ', rel_error(out, correct_out))
```

```
Testing relu_forward function:
difference:  4.999999798022158e-08
```

5 ReLU activation: backward

Now implement the backward pass for the ReLU activation function in the `relu_backward` function and test your implementation using numeric gradient checking:

[8]:
```
np.random.seed(231)
x = np.random.randn(10, 10)
dout = np.random.randn(*x.shape)

dx_num = eval_numerical_gradient_array(lambda x: relu_forward(x)[0], x, dout)

_, cache = relu_forward(x)
dx = relu_backward(dout, cache)

# The error should be on the order of e-12
print('Testing relu_backward function:')
print('dx error: ', rel_error(dx_num, dx))
```

```
Testing relu_backward function:
dx error:  3.2756349136310288e-12
```

5.1 Inline Question 1:

We've only asked you to implement ReLU, but there are a number of different activation functions that one could use in neural networks, each with its pros and cons. In particular, an issue commonly seen with activation functions is getting zero (or close to zero) gradient flow during backpropagation. Which of the following activation functions have this problem? If you consider these functions in the one dimensional case, what types of input would lead to this behaviour? 1. Sigmoid 2. ReLU 3. Leaky ReLU

5.2 Answer: 1

Sigmoid function saturates for x with very large absolute value, and the gradient is getting closer to 0 as the absolute value of x increases.

6 "Sandwich" layers

There are some common patterns of layers that are frequently used in neural nets. For example, affine layers are frequently followed by a ReLU nonlinearity. To make these common patterns easy, we define several convenience layers in the file `cs231n/layer_utils.py`.

For now take a look at the `affine_relu_forward` and `affine_relu_backward` functions, and run the following to numerically gradient check the backward pass:

[9]:
```python
from cs231n.layer_utils import affine_relu_forward, affine_relu_backward
np.random.seed(231)
x = np.random.randn(2, 3, 4)
w = np.random.randn(12, 10)
b = np.random.randn(10)
dout = np.random.randn(2, 10)

out, cache = affine_relu_forward(x, w, b)
dx, dw, db = affine_relu_backward(dout, cache)

dx_num = eval_numerical_gradient_array(lambda x: affine_relu_forward(x, w,␣
 ↪b)[0], x, dout)
dw_num = eval_numerical_gradient_array(lambda w: affine_relu_forward(x, w,␣
 ↪b)[0], w, dout)
db_num = eval_numerical_gradient_array(lambda b: affine_relu_forward(x, w,␣
 ↪b)[0], b, dout)

# Relative error should be around e-10 or less
print('Testing affine_relu_forward and affine_relu_backward:')
print('dx error: ', rel_error(dx_num, dx))
print('dw error: ', rel_error(dw_num, dw))
print('db error: ', rel_error(db_num, db))
```

```
Testing affine_relu_forward and affine_relu_backward:
dx error:  2.299579177309368e-11
dw error:  8.162011105764925e-11
db error:  7.826724021458994e-12
```

7 Loss layers: Softmax and SVM

Now implement the loss and gradient for softmax and SVM in the `softmax_loss` and `svm_loss` function in `cs231n/layers.py`. These should be similar to what you implemented in `cs231n/classifiers/softmax.py` and `cs231n/classifiers/linear_svm.py`.

You can make sure that the implementations are correct by running the following:

[10]:
```python
np.random.seed(231)
num_classes, num_inputs = 10, 50
x = 0.001 * np.random.randn(num_inputs, num_classes)
y = np.random.randint(num_classes, size=num_inputs)

dx_num = eval_numerical_gradient(lambda x: svm_loss(x, y)[0], x, verbose=False)
loss, dx = svm_loss(x, y)

# Test svm_loss function. Loss should be around 9 and dx error should be around␣
 ↪the order of e-9
print('Testing svm_loss:')
print('loss: ', loss)
print('dx error: ', rel_error(dx_num, dx))

dx_num = eval_numerical_gradient(lambda x: softmax_loss(x, y)[0], x,␣
 ↪verbose=False)
loss, dx = softmax_loss(x, y)

# Test softmax_loss function. Loss should be close to 2.3 and dx error should␣
 ↪be around e-8

# The desired error can be reached when we delete the line 212 in layers.py.
# However, the line 212 is added for numerical stability in further training␣
 ↪processes.
# This doesn't affect the overall leraning process.

print('\nTesting softmax_loss:')
print('loss: ', loss)
print('dx error: ', rel_error(dx_num, dx))
```

```
Testing svm_loss:
loss:  8.999602749096233
dx error:  1.4021566006651672e-09

Testing softmax_loss:
loss:  2.302545844500738
dx error:  0.3333335561857877
```

8 Two-layer network

Open the file `cs231n/classifiers/fc_net.py` and complete the implementation of the `TwoLayerNet` class. Read through it to make sure you understand the API. You can run the cell below to test your implementation.

[11]:
```
np.random.seed(231)
N, D, H, C = 3, 5, 50, 7
X = np.random.randn(N, D)
y = np.random.randint(C, size=N)

std = 1e-3
model = TwoLayerNet(input_dim=D, hidden_dim=H, num_classes=C, weight_scale=std)

print('Testing initialization ... ')
W1_std = abs(model.params['W1'].std() - std)
b1 = model.params['b1']
W2_std = abs(model.params['W2'].std() - std)
b2 = model.params['b2']
assert W1_std < std / 10, 'First layer weights do not seem right'
assert np.all(b1 == 0), 'First layer biases do not seem right'
assert W2_std < std / 10, 'Second layer weights do not seem right'
assert np.all(b2 == 0), 'Second layer biases do not seem right'

print('Testing test-time forward pass ... ')
model.params['W1'] = np.linspace(-0.7, 0.3, num=D*H).reshape(D, H)
model.params['b1'] = np.linspace(-0.1, 0.9, num=H)
model.params['W2'] = np.linspace(-0.3, 0.4, num=H*C).reshape(H, C)
model.params['b2'] = np.linspace(-0.9, 0.1, num=C)
X = np.linspace(-5.5, 4.5, num=N*D).reshape(D, N).T
scores = model.loss(X)
correct_scores = np.asarray(
  [[11.53165108,  12.2917344,   13.05181771,  13.81190102,  14.57198434, 15.
↪33206765,  16.09215096],
   [12.05769098,  12.74614105,  13.43459113,  14.1230412,   14.81149128, 15.
↪49994135,  16.18839143],
   [12.58373087,  13.20054771,  13.81736455,  14.43418138,  15.05099822, 15.
↪66781506,  16.2846319 ]])
scores_diff = np.abs(scores - correct_scores).sum()
assert scores_diff < 1e-6, 'Problem with test-time forward pass'

print('Testing training loss (no regularization)')
y = np.asarray([0, 5, 1])
loss, grads = model.loss(X, y)
correct_loss = 3.4702243556
assert abs(loss - correct_loss) < 1e-10, 'Problem with training-time loss'

model.reg = 1.0
loss, grads = model.loss(X, y)
correct_loss = 26.5948426952
assert abs(loss - correct_loss) < 1e-10, 'Problem with regularization loss'

# Errors should be around e-7 or less
for reg in [0.0, 0.7]:
  print('Running numeric gradient check with reg = ', reg)
  model.reg = reg
  loss, grads = model.loss(X, y)

  for name in sorted(grads):
    f = lambda _: model.loss(X, y)[0]
    grad_num = eval_numerical_gradient(f, model.params[name], verbose=False)
    print('%s relative error: %.2e' % (name, rel_error(grad_num, grads[name])))
```

```
Testing initialization ...
Testing test-time forward pass ...
Testing training loss (no regularization)
Running numeric gradient check with reg =  0.0
W1 relative error: 1.83e-08
```

```
W2 relative error: 3.12e-10
b1 relative error: 9.83e-09
b2 relative error: 4.33e-10
Running numeric gradient check with reg =  0.7
W1 relative error: 2.53e-07
W2 relative error: 7.98e-08
b1 relative error: 1.35e-08
b2 relative error: 7.76e-10
```

9 Solver

Open the file `cs231n/solver.py` and read through it to familiarize yourself with the API. You also need to imeplement the `sgd` function in `cs231n/optim.py`. After doing so, use a `Solver` instance to train a `TwoLayerNet` that achieves about 36% accuracy on the validation set.

[12]:
```
input_size = 32 * 32 * 3
hidden_size = 50
num_classes = 10
model = TwoLayerNet(input_size, hidden_size, num_classes)
solver = None

##############################################################################
# TODO: Use a Solver instance to train a TwoLayerNet that achieves about 36% #
# accuracy on the validation set.                                            #
##############################################################################
# *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****

solver = Solver(model, data,
                update_rule='sgd',
                optim_config={
                  'learning_rate': 1e-3,
                },
                lr_decay=0.95,
                num_epochs=10, batch_size=100,
                print_every=100)
solver.train()

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

```
(Iteration 1 / 4900) loss: 2.300089
(Epoch 0 / 10) train acc: 0.171000; val_acc: 0.170000
(Iteration 101 / 4900) loss: 1.782419
(Iteration 201 / 4900) loss: 1.803466
(Iteration 301 / 4900) loss: 1.712676
(Iteration 401 / 4900) loss: 1.693946
```

```
(Epoch 1 / 10) train acc: 0.399000; val_acc: 0.428000
(Iteration 501 / 4900) loss: 1.711237
(Iteration 601 / 4900) loss: 1.443683
(Iteration 701 / 4900) loss: 1.574904
(Iteration 801 / 4900) loss: 1.526751
(Iteration 901 / 4900) loss: 1.352554
(Epoch 2 / 10) train acc: 0.463000; val_acc: 0.443000
(Iteration 1001 / 4900) loss: 1.340071
(Iteration 1101 / 4900) loss: 1.386843
(Iteration 1201 / 4900) loss: 1.489919
(Iteration 1301 / 4900) loss: 1.353077
(Iteration 1401 / 4900) loss: 1.467951
(Epoch 3 / 10) train acc: 0.477000; val_acc: 0.430000
(Iteration 1501 / 4900) loss: 1.337133
(Iteration 1601 / 4900) loss: 1.426819
(Iteration 1701 / 4900) loss: 1.348675
(Iteration 1801 / 4900) loss: 1.412626
(Iteration 1901 / 4900) loss: 1.354764
(Epoch 4 / 10) train acc: 0.497000; val_acc: 0.467000
(Iteration 2001 / 4900) loss: 1.422221
(Iteration 2101 / 4900) loss: 1.360665
(Iteration 2201 / 4900) loss: 1.546539
(Iteration 2301 / 4900) loss: 1.196704
(Iteration 2401 / 4900) loss: 1.401958
(Epoch 5 / 10) train acc: 0.508000; val_acc: 0.483000
(Iteration 2501 / 4900) loss: 1.243567
(Iteration 2601 / 4900) loss: 1.513808
(Iteration 2701 / 4900) loss: 1.266416
(Iteration 2801 / 4900) loss: 1.485956
(Iteration 2901 / 4900) loss: 1.049706
(Epoch 6 / 10) train acc: 0.533000; val_acc: 0.492000
(Iteration 3001 / 4900) loss: 1.264002
(Iteration 3101 / 4900) loss: 1.184786
(Iteration 3201 / 4900) loss: 1.231162
(Iteration 3301 / 4900) loss: 1.353725
(Iteration 3401 / 4900) loss: 1.217830
(Epoch 7 / 10) train acc: 0.545000; val_acc: 0.489000
(Iteration 3501 / 4900) loss: 1.446003
(Iteration 3601 / 4900) loss: 1.152495
(Iteration 3701 / 4900) loss: 1.421970
(Iteration 3801 / 4900) loss: 1.222752
(Iteration 3901 / 4900) loss: 1.213468
(Epoch 8 / 10) train acc: 0.546000; val_acc: 0.474000
(Iteration 4001 / 4900) loss: 1.187435
(Iteration 4101 / 4900) loss: 1.284799
(Iteration 4201 / 4900) loss: 1.135251
(Iteration 4301 / 4900) loss: 1.212217
(Iteration 4401 / 4900) loss: 1.213544
```

```
(Epoch 9 / 10) train acc: 0.586000; val_acc: 0.486000
(Iteration 4501 / 4900) loss: 1.306174
(Iteration 4601 / 4900) loss: 1.213528
(Iteration 4701 / 4900) loss: 1.220260
(Iteration 4801 / 4900) loss: 1.233231
(Epoch 10 / 10) train acc: 0.545000; val_acc: 0.483000
```

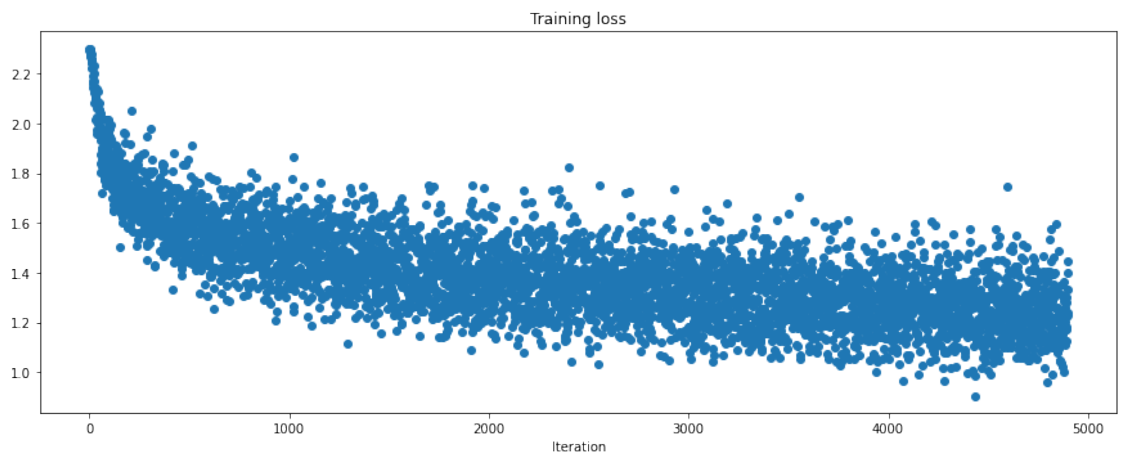
10 Debug the training

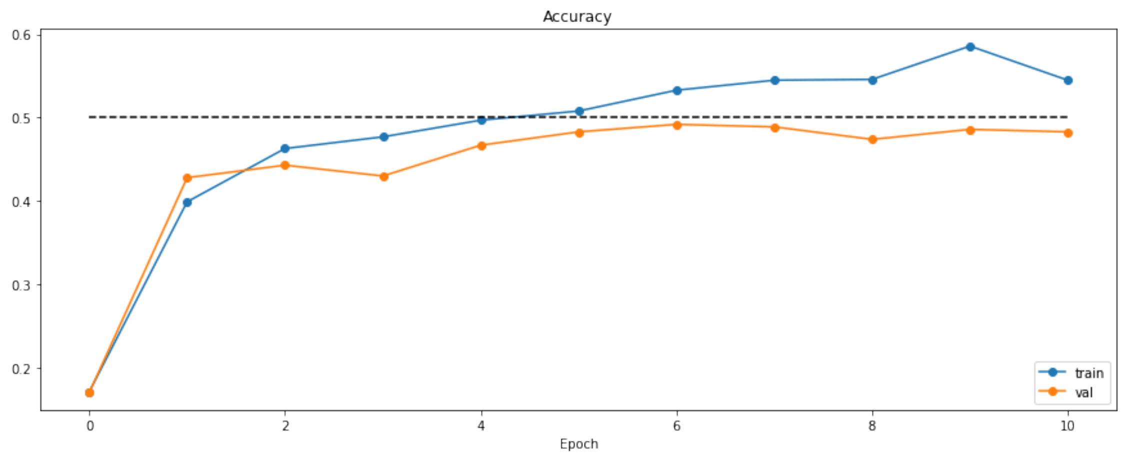
With the default parameters we provided above, you should get a validation accuracy of about 0.36 on the validation set. This isn't very good.

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.

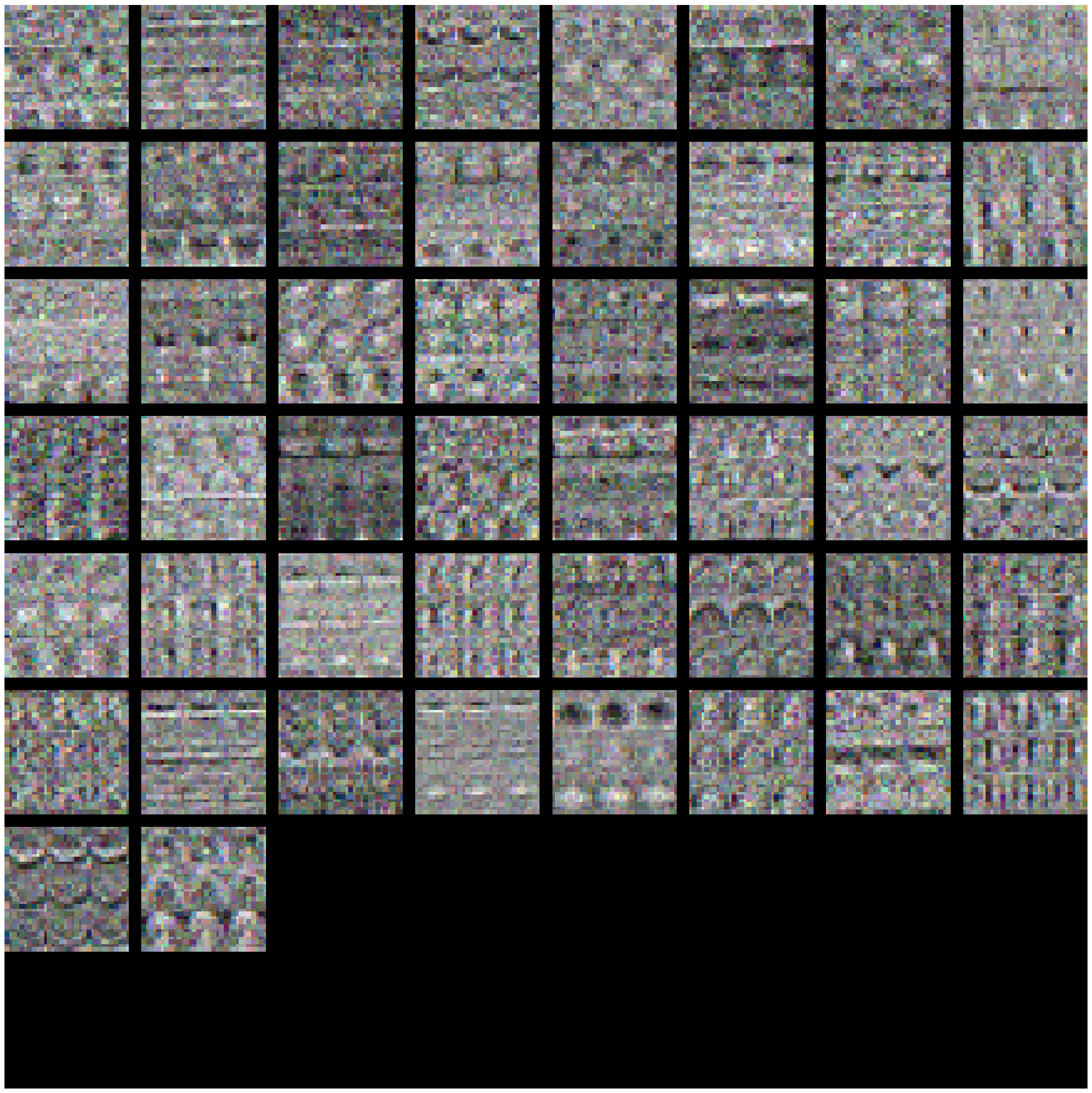
Another strategy is to visualize the weights that were learned in the first layer of the network. In most neural networks trained on visual data, the first layer weights typically show some visible structure when visualized.

[13]:
```
# Run this cell to visualize training loss and train / val accuracy

plt.subplot(2, 1, 1)
plt.title('Training loss')
plt.plot(solver.loss_history, 'o')
plt.xlabel('Iteration')

plt.subplot(2, 1, 2)
plt.title('Accuracy')
plt.plot(solver.train_acc_history, '-o', label='train')
plt.plot(solver.val_acc_history, '-o', label='val')
plt.plot([0.5] * len(solver.val_acc_history), 'k--')
plt.xlabel('Epoch')
plt.legend(loc='lower right')
plt.gcf().set_size_inches(15, 12)
plt.show()
```





[14]:
```python
from cs231n.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(model)
```



11 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 (52% could serve as a reference), 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.).

[15]:
```
best_model = None


#################################################################################
# TODO: Tune hyperparameters using the validation set. Store your best trained ␣
 ↪#
# model in best_model.                                                          ␣
 ↪#
#                                                                               ␣
 ↪#
# To help debug your network, it may help to use visualizations similar to the ␣
 ↪#
# ones we used above; these visualizations will have significant qualitative   ␣
 ↪#
# differences from the ones we saw above for the poorly tuned network.          ␣
 ↪#
#                                                                               ␣
 ↪#
# Tweaking hyperparameters by hand can be fun, but you might find it useful to ␣
 ↪#
# write code to sweep through possible combinations of hyperparameters          ␣
 ↪#
# automatically like we did on thexs previous exercises.                        ␣
 ↪#
#################################################################################
# *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****

best_solver = None
num_trials = 5

for i in range(num_trials):

  _hidden_size = np.random.randint(50, 200)
  _reg = np.random.uniform(0, 3)
  _learning_rate = 10**(-np.random.uniform(3, 4.3))
  _lr_decay = np.random.uniform(0.9, 1)
  _num_epochs = 10
  if _learning_rate < 10**-4:
    _num_epochs = 12

  model = TwoLayerNet(hidden_dim = _hidden_size, reg = _reg)
  solver = Solver(model, data,
                update_rule='sgd',
                optim_config={
                  'learning_rate': _learning_rate,
                },
                lr_decay=_lr_decay,
                num_epochs=_num_epochs,
                batch_size=100,
                print_every=100)

  print("Start computing trial {}/{}...".format(i, num_trials))
  print("hidden size: {}, reg: {}, lr: {}, lr_decay: {}".format(_hidden_size,␣
↪_reg, _learning_rate, _lr_decay))

  solver.train()

  if best_model == None:
    best_model = model
    best_solver = solver

  else:
    if solver.best_val_acc > best_solver.best_val_acc:
      best_model = model
      best_solver= solver
      print("Best model updated! Current best_val_acc = {}".format(best_solver.
↪best_val_acc))

    else:
      print("Best model not updated! Current best_val_acc = {}".
↪format(best_solver.best_val_acc))


# Run this cell to visualize training loss and train / val accuracy

plt.subplot(2, 1, 1)
plt.title('Training loss')
plt.plot(best_solver.loss_history, 'o')
plt.xlabel('Iteration')

plt.subplot(2, 1, 2)
plt.title('Accuracy')
plt.plot(best_solver.train_acc_history, '-o', label='train')
plt.plot(best_solver.val_acc_history, '-o', label='val')
plt.plot([0.5] * len(best_solver.val_acc_history), 'k--')
plt.xlabel('Epoch')
plt.legend(loc='lower right')
plt.gcf().set_size_inches(15, 12)
plt.show()

# 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(model)

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

```
Start computing trial 0/5...
hidden size: 67, reg: 0.8726721924716342, lr: 0.00029145778926922285, lr_decay:
0.9207477790701888
(Iteration 1 / 4900) loss: 2.392034
(Epoch 0 / 10) train acc: 0.111000; val_acc: 0.117000
(Iteration 101 / 4900) loss: 2.167018
(Iteration 201 / 4900) loss: 2.025364
(Iteration 301 / 4900) loss: 1.936633
(Iteration 401 / 4900) loss: 1.926664
(Epoch 1 / 10) train acc: 0.411000; val_acc: 0.388000
(Iteration 501 / 4900) loss: 1.875830
(Iteration 601 / 4900) loss: 1.738151
(Iteration 701 / 4900) loss: 1.871370
(Iteration 801 / 4900) loss: 1.781361
(Iteration 901 / 4900) loss: 1.815897
(Epoch 2 / 10) train acc: 0.431000; val_acc: 0.434000
(Iteration 1001 / 4900) loss: 1.719220
(Iteration 1101 / 4900) loss: 1.911048
(Iteration 1201 / 4900) loss: 1.678416
(Iteration 1301 / 4900) loss: 1.827024
(Iteration 1401 / 4900) loss: 1.709019
(Epoch 3 / 10) train acc: 0.479000; val_acc: 0.457000
(Iteration 1501 / 4900) loss: 1.622489
(Iteration 1601 / 4900) loss: 1.441175
(Iteration 1701 / 4900) loss: 1.457415
```

```
(Iteration 1801 / 4900) loss: 1.664764
(Iteration 1901 / 4900) loss: 1.576585
(Epoch 4 / 10) train acc: 0.481000; val_acc: 0.467000
(Iteration 2001 / 4900) loss: 1.477055
(Iteration 2101 / 4900) loss: 1.523519
(Iteration 2201 / 4900) loss: 1.468488
(Iteration 2301 / 4900) loss: 1.625306
(Iteration 2401 / 4900) loss: 1.571530
(Epoch 5 / 10) train acc: 0.473000; val_acc: 0.466000
(Iteration 2501 / 4900) loss: 1.693640
(Iteration 2601 / 4900) loss: 1.549521
(Iteration 2701 / 4900) loss: 1.626045
(Iteration 2801 / 4900) loss: 1.476333
(Iteration 2901 / 4900) loss: 1.610074
(Epoch 6 / 10) train acc: 0.503000; val_acc: 0.474000
(Iteration 3001 / 4900) loss: 1.507888
(Iteration 3101 / 4900) loss: 1.383574
(Iteration 3201 / 4900) loss: 1.659455
(Iteration 3301 / 4900) loss: 1.549482
(Iteration 3401 / 4900) loss: 1.574629
(Epoch 7 / 10) train acc: 0.512000; val_acc: 0.487000
(Iteration 3501 / 4900) loss: 1.551348
(Iteration 3601 / 4900) loss: 1.479303
(Iteration 3701 / 4900) loss: 1.436317
(Iteration 3801 / 4900) loss: 1.459550
(Iteration 3901 / 4900) loss: 1.498808
(Epoch 8 / 10) train acc: 0.506000; val_acc: 0.486000
(Iteration 4001 / 4900) loss: 1.469148
(Iteration 4101 / 4900) loss: 1.435611
(Iteration 4201 / 4900) loss: 1.485736
(Iteration 4301 / 4900) loss: 1.601615
(Iteration 4401 / 4900) loss: 1.555688
(Epoch 9 / 10) train acc: 0.535000; val_acc: 0.501000
(Iteration 4501 / 4900) loss: 1.520582
(Iteration 4601 / 4900) loss: 1.516518
(Iteration 4701 / 4900) loss: 1.478080
(Iteration 4801 / 4900) loss: 1.644383
(Epoch 10 / 10) train acc: 0.529000; val_acc: 0.500000
Start computing trial 1/5...
hidden size: 135, reg: 2.8232503683088734, lr: 0.00010932188191168741, lr_decay:
0.945336095721318
(Iteration 1 / 4900) loss: 2.883195
(Epoch 0 / 10) train acc: 0.121000; val_acc: 0.113000
(Iteration 101 / 4900) loss: 2.770183
(Iteration 201 / 4900) loss: 2.701246
(Iteration 301 / 4900) loss: 2.525100
(Iteration 401 / 4900) loss: 2.414792
(Epoch 1 / 10) train acc: 0.330000; val_acc: 0.321000
```

```
(Iteration 501 / 4900) loss: 2.379728
(Iteration 601 / 4900) loss: 2.230713
(Iteration 701 / 4900) loss: 2.285499
(Iteration 801 / 4900) loss: 2.217190
(Iteration 901 / 4900) loss: 2.095318
(Epoch 2 / 10) train acc: 0.375000; val_acc: 0.378000
(Iteration 1001 / 4900) loss: 2.098312
(Iteration 1101 / 4900) loss: 2.163646
(Iteration 1201 / 4900) loss: 2.098597
(Iteration 1301 / 4900) loss: 2.024961
(Iteration 1401 / 4900) loss: 1.936072
(Epoch 3 / 10) train acc: 0.390000; val_acc: 0.403000
(Iteration 1501 / 4900) loss: 2.051227
(Iteration 1601 / 4900) loss: 2.059418
(Iteration 1701 / 4900) loss: 2.078292
(Iteration 1801 / 4900) loss: 1.992452
(Iteration 1901 / 4900) loss: 2.017170
(Epoch 4 / 10) train acc: 0.388000; val_acc: 0.405000
(Iteration 2001 / 4900) loss: 1.958471
(Iteration 2101 / 4900) loss: 1.968951
(Iteration 2201 / 4900) loss: 2.021852
(Iteration 2301 / 4900) loss: 1.920427
(Iteration 2401 / 4900) loss: 1.837431
(Epoch 5 / 10) train acc: 0.413000; val_acc: 0.430000
(Iteration 2501 / 4900) loss: 1.797813
(Iteration 2601 / 4900) loss: 2.050298
(Iteration 2701 / 4900) loss: 1.689238
(Iteration 2801 / 4900) loss: 1.864425
(Iteration 2901 / 4900) loss: 1.869501
(Epoch 6 / 10) train acc: 0.423000; val_acc: 0.451000
(Iteration 3001 / 4900) loss: 1.844650
(Iteration 3101 / 4900) loss: 2.017368
(Iteration 3201 / 4900) loss: 1.886630
(Iteration 3301 / 4900) loss: 1.858581
(Iteration 3401 / 4900) loss: 1.805974
(Epoch 7 / 10) train acc: 0.441000; val_acc: 0.450000
(Iteration 3501 / 4900) loss: 1.751710
(Iteration 3601 / 4900) loss: 1.816931
(Iteration 3701 / 4900) loss: 1.858710
(Iteration 3801 / 4900) loss: 1.652978
(Iteration 3901 / 4900) loss: 1.773891
(Epoch 8 / 10) train acc: 0.453000; val_acc: 0.443000
(Iteration 4001 / 4900) loss: 1.850342
(Iteration 4101 / 4900) loss: 1.768414
(Iteration 4201 / 4900) loss: 1.936375
(Iteration 4301 / 4900) loss: 1.657677
(Iteration 4401 / 4900) loss: 1.751190
(Epoch 9 / 10) train acc: 0.436000; val_acc: 0.461000
```

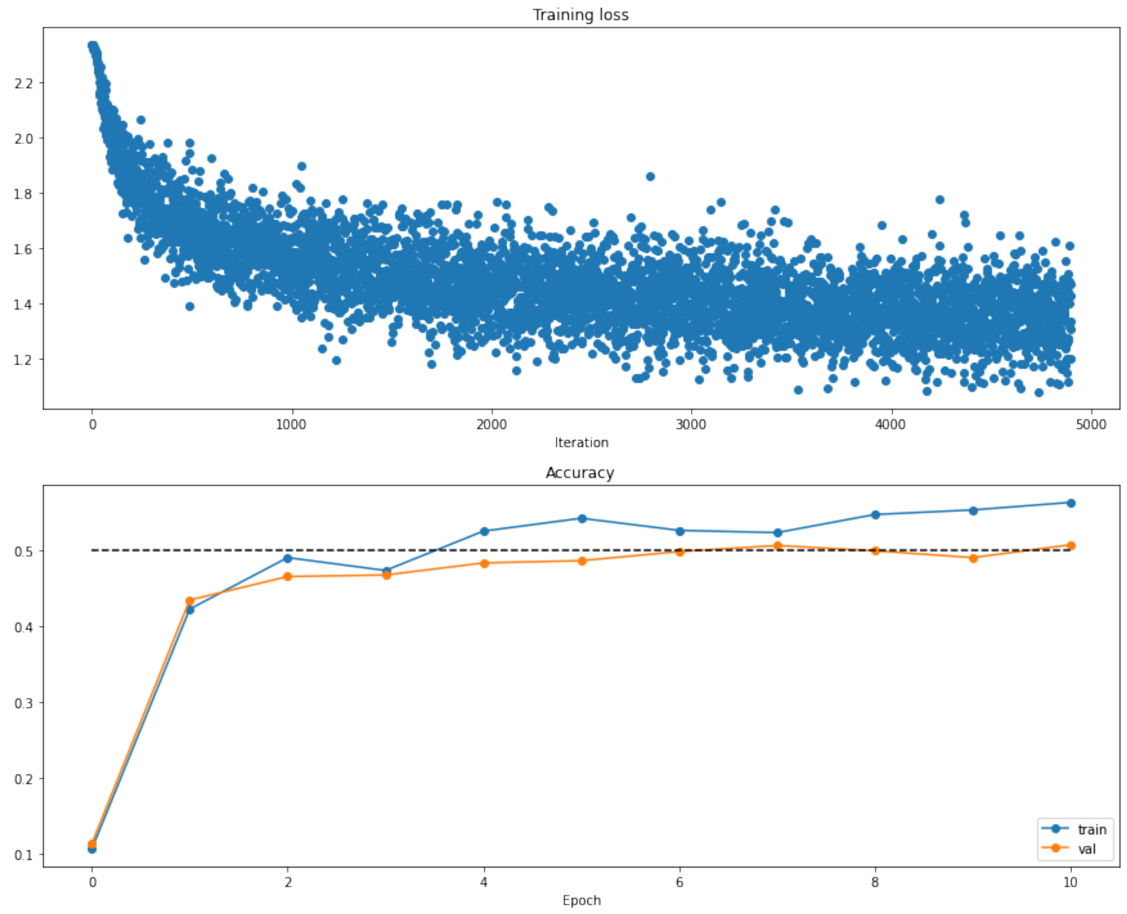
```
(Iteration 4501 / 4900) loss: 1.773541
(Iteration 4601 / 4900) loss: 1.648886
(Iteration 4701 / 4900) loss: 1.796288
(Iteration 4801 / 4900) loss: 1.659725
(Epoch 10 / 10) train acc: 0.440000; val_acc: 0.460000
Best model not updated! Current best_val_acc = 0.501
Start computing trial 2/5...
hidden size: 98, reg: 0.02187354501365435, lr: 0.00010728566554394516, lr_decay:
0.9909023754243242
(Iteration 1 / 4900) loss: 2.311067
(Epoch 0 / 10) train acc: 0.085000; val_acc: 0.094000
(Iteration 101 / 4900) loss: 2.257329
(Iteration 201 / 4900) loss: 2.093860
(Iteration 301 / 4900) loss: 2.173822
(Iteration 401 / 4900) loss: 1.984894
(Epoch 1 / 10) train acc: 0.328000; val_acc: 0.316000
(Iteration 501 / 4900) loss: 1.821464
(Iteration 601 / 4900) loss: 1.892181
(Iteration 701 / 4900) loss: 1.872646
(Iteration 801 / 4900) loss: 1.967298
(Iteration 901 / 4900) loss: 1.714240
(Epoch 2 / 10) train acc: 0.355000; val_acc: 0.375000
(Iteration 1001 / 4900) loss: 1.720773
(Iteration 1101 / 4900) loss: 1.981680
(Iteration 1201 / 4900) loss: 1.892650
(Iteration 1301 / 4900) loss: 1.683674
(Iteration 1401 / 4900) loss: 1.753329
(Epoch 3 / 10) train acc: 0.401000; val_acc: 0.412000
(Iteration 1501 / 4900) loss: 1.688490
(Iteration 1601 / 4900) loss: 1.943572
(Iteration 1701 / 4900) loss: 1.602861
(Iteration 1801 / 4900) loss: 1.553358
(Iteration 1901 / 4900) loss: 1.642333
(Epoch 4 / 10) train acc: 0.440000; val_acc: 0.426000
(Iteration 2001 / 4900) loss: 1.562107
(Iteration 2101 / 4900) loss: 1.535876
(Iteration 2201 / 4900) loss: 1.623910
(Iteration 2301 / 4900) loss: 1.478674
(Iteration 2401 / 4900) loss: 1.698338
(Epoch 5 / 10) train acc: 0.463000; val_acc: 0.452000
(Iteration 2501 / 4900) loss: 1.508185
(Iteration 2601 / 4900) loss: 1.789541
(Iteration 2701 / 4900) loss: 1.567672
(Iteration 2801 / 4900) loss: 1.564676
(Iteration 2901 / 4900) loss: 1.570802
(Epoch 6 / 10) train acc: 0.458000; val_acc: 0.464000
(Iteration 3001 / 4900) loss: 1.410037
(Iteration 3101 / 4900) loss: 1.555088
```

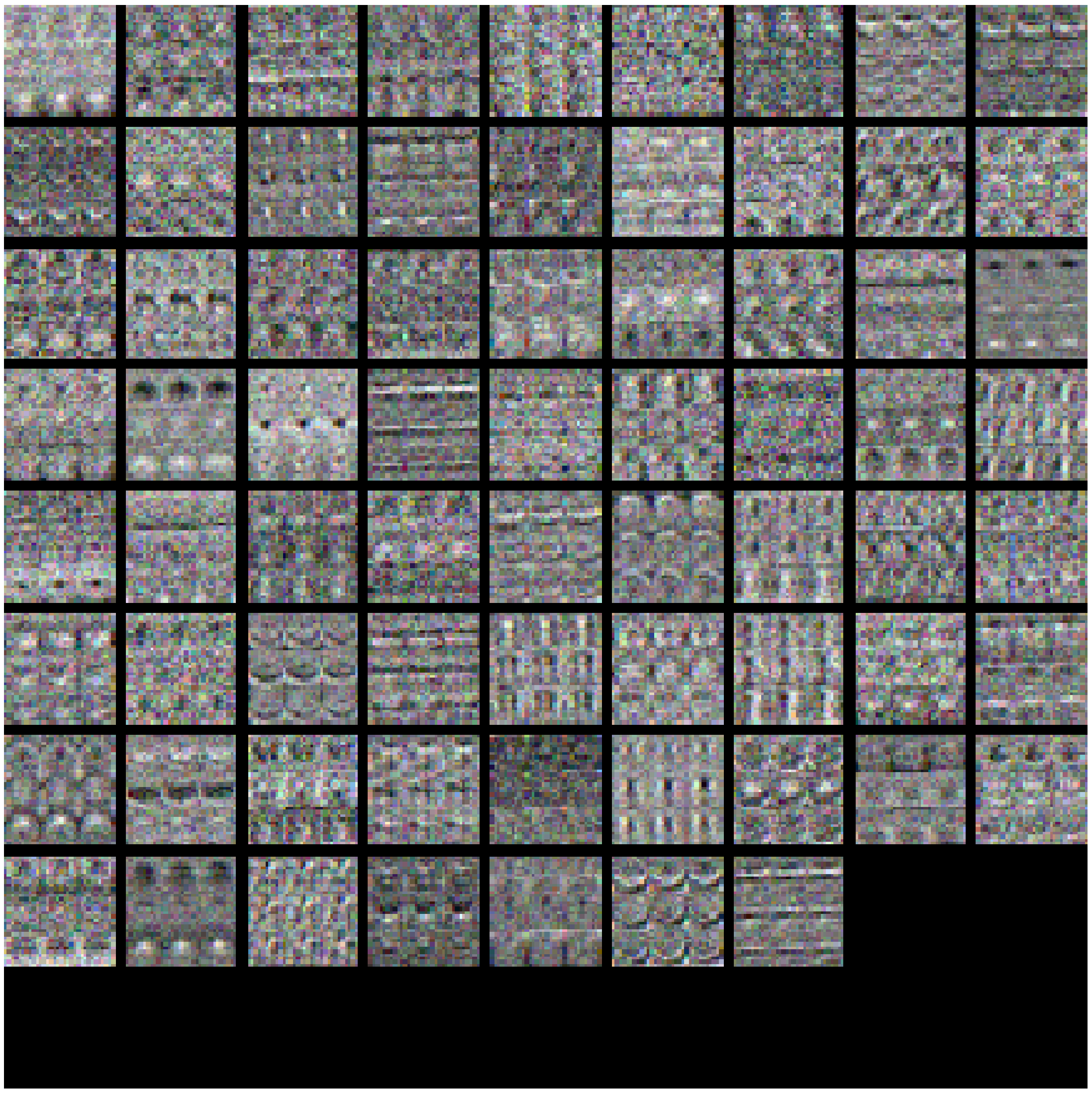
```
(Iteration 3201 / 4900) loss: 1.275938
(Iteration 3301 / 4900) loss: 1.575537
(Iteration 3401 / 4900) loss: 1.458227
(Epoch 7 / 10) train acc: 0.450000; val_acc: 0.456000
(Iteration 3501 / 4900) loss: 1.406650
(Iteration 3601 / 4900) loss: 1.610227
(Iteration 3701 / 4900) loss: 1.600525
(Iteration 3801 / 4900) loss: 1.366303
(Iteration 3901 / 4900) loss: 1.508926
(Epoch 8 / 10) train acc: 0.484000; val_acc: 0.463000
(Iteration 4001 / 4900) loss: 1.588167
(Iteration 4101 / 4900) loss: 1.440300
(Iteration 4201 / 4900) loss: 1.507272
(Iteration 4301 / 4900) loss: 1.291458
(Iteration 4401 / 4900) loss: 1.307732
(Epoch 9 / 10) train acc: 0.499000; val_acc: 0.459000
(Iteration 4501 / 4900) loss: 1.329901
(Iteration 4601 / 4900) loss: 1.372066
(Iteration 4701 / 4900) loss: 1.556712
(Iteration 4801 / 4900) loss: 1.354775
(Epoch 10 / 10) train acc: 0.497000; val_acc: 0.473000
Best model not updated! Current best_val_acc = 0.501
Start computing trial 3/5...
hidden size: 50, reg: 0.23208537941678364, lr: 0.00014979368580802087, lr_decay:
0.9556454350933291
(Iteration 1 / 4900) loss: 2.319526
(Epoch 0 / 10) train acc: 0.103000; val_acc: 0.110000
(Iteration 101 / 4900) loss: 2.192885
(Iteration 201 / 4900) loss: 2.158711
(Iteration 301 / 4900) loss: 1.962620
(Iteration 401 / 4900) loss: 1.977064
(Epoch 1 / 10) train acc: 0.344000; val_acc: 0.339000
(Iteration 501 / 4900) loss: 1.967700
(Iteration 601 / 4900) loss: 1.977386
(Iteration 701 / 4900) loss: 1.827049
(Iteration 801 / 4900) loss: 1.668663
(Iteration 901 / 4900) loss: 1.865377
(Epoch 2 / 10) train acc: 0.405000; val_acc: 0.380000
(Iteration 1001 / 4900) loss: 1.722781
(Iteration 1101 / 4900) loss: 1.604354
(Iteration 1201 / 4900) loss: 1.571469
(Iteration 1301 / 4900) loss: 1.689815
(Iteration 1401 / 4900) loss: 1.728840
(Epoch 3 / 10) train acc: 0.442000; val_acc: 0.435000
(Iteration 1501 / 4900) loss: 1.795195
(Iteration 1601 / 4900) loss: 1.704467
(Iteration 1701 / 4900) loss: 1.455069
(Iteration 1801 / 4900) loss: 1.628481
```

```
(Iteration 1901 / 4900) loss: 1.647591
(Epoch 4 / 10) train acc: 0.438000; val_acc: 0.445000
(Iteration 2001 / 4900) loss: 1.364263
(Iteration 2101 / 4900) loss: 1.671652
(Iteration 2201 / 4900) loss: 1.602959
(Iteration 2301 / 4900) loss: 1.587719
(Iteration 2401 / 4900) loss: 1.501681
(Epoch 5 / 10) train acc: 0.468000; val_acc: 0.456000
(Iteration 2501 / 4900) loss: 1.386618
(Iteration 2601 / 4900) loss: 1.588856
(Iteration 2701 / 4900) loss: 1.754708
(Iteration 2801 / 4900) loss: 1.488514
(Iteration 2901 / 4900) loss: 1.648110
(Epoch 6 / 10) train acc: 0.451000; val_acc: 0.465000
(Iteration 3001 / 4900) loss: 1.453123
(Iteration 3101 / 4900) loss: 1.579634
(Iteration 3201 / 4900) loss: 1.511488
(Iteration 3301 / 4900) loss: 1.580036
(Iteration 3401 / 4900) loss: 1.521179
(Epoch 7 / 10) train acc: 0.468000; val_acc: 0.474000
(Iteration 3501 / 4900) loss: 1.657842
(Iteration 3601 / 4900) loss: 1.446584
(Iteration 3701 / 4900) loss: 1.630573
(Iteration 3801 / 4900) loss: 1.495977
(Iteration 3901 / 4900) loss: 1.529816
(Epoch 8 / 10) train acc: 0.447000; val_acc: 0.471000
(Iteration 4001 / 4900) loss: 1.504086
(Iteration 4101 / 4900) loss: 1.486339
(Iteration 4201 / 4900) loss: 1.328748
(Iteration 4301 / 4900) loss: 1.447076
(Iteration 4401 / 4900) loss: 1.358363
(Epoch 9 / 10) train acc: 0.476000; val_acc: 0.472000
(Iteration 4501 / 4900) loss: 1.286458
(Iteration 4601 / 4900) loss: 1.337359
(Iteration 4701 / 4900) loss: 1.569713
(Iteration 4801 / 4900) loss: 1.383756
(Epoch 10 / 10) train acc: 0.498000; val_acc: 0.476000
Best model not updated! Current best_val_acc = 0.501
Start computing trial 4/5...
hidden size: 70, reg: 0.29047325084402575, lr: 0.00048620185617124183, lr_decay:
0.9070481448917642
(Iteration 1 / 4900) loss: 2.334937
(Epoch 0 / 10) train acc: 0.106000; val_acc: 0.113000
(Iteration 101 / 4900) loss: 1.947701
(Iteration 201 / 4900) loss: 1.791225
(Iteration 301 / 4900) loss: 1.658607
(Iteration 401 / 4900) loss: 1.659293
(Epoch 1 / 10) train acc: 0.423000; val_acc: 0.435000
```

```
(Iteration 501 / 4900) loss: 1.882797
(Iteration 601 / 4900) loss: 1.579984
(Iteration 701 / 4900) loss: 1.628963
(Iteration 801 / 4900) loss: 1.426685
(Iteration 901 / 4900) loss: 1.616596
(Epoch 2 / 10) train acc: 0.491000; val_acc: 0.466000
(Iteration 1001 / 4900) loss: 1.673222
(Iteration 1101 / 4900) loss: 1.568182
(Iteration 1201 / 4900) loss: 1.653492
(Iteration 1301 / 4900) loss: 1.537675
(Iteration 1401 / 4900) loss: 1.599937
(Epoch 3 / 10) train acc: 0.474000; val_acc: 0.468000
(Iteration 1501 / 4900) loss: 1.626723
(Iteration 1601 / 4900) loss: 1.565988
(Iteration 1701 / 4900) loss: 1.506678
(Iteration 1801 / 4900) loss: 1.518635
(Iteration 1901 / 4900) loss: 1.583540
(Epoch 4 / 10) train acc: 0.526000; val_acc: 0.484000
(Iteration 2001 / 4900) loss: 1.482676
(Iteration 2101 / 4900) loss: 1.483021
(Iteration 2201 / 4900) loss: 1.320651
(Iteration 2301 / 4900) loss: 1.525165
(Iteration 2401 / 4900) loss: 1.254804
(Epoch 5 / 10) train acc: 0.543000; val_acc: 0.487000
(Iteration 2501 / 4900) loss: 1.525289
(Iteration 2601 / 4900) loss: 1.522729
(Iteration 2701 / 4900) loss: 1.539071
(Iteration 2801 / 4900) loss: 1.423916
(Iteration 2901 / 4900) loss: 1.422288
(Epoch 6 / 10) train acc: 0.527000; val_acc: 0.499000
(Iteration 3001 / 4900) loss: 1.303964
(Iteration 3101 / 4900) loss: 1.223734
(Iteration 3201 / 4900) loss: 1.397342
(Iteration 3301 / 4900) loss: 1.539742
(Iteration 3401 / 4900) loss: 1.492786
(Epoch 7 / 10) train acc: 0.524000; val_acc: 0.507000
(Iteration 3501 / 4900) loss: 1.327961
(Iteration 3601 / 4900) loss: 1.480856
(Iteration 3701 / 4900) loss: 1.314604
(Iteration 3801 / 4900) loss: 1.252915
(Iteration 3901 / 4900) loss: 1.336912
(Epoch 8 / 10) train acc: 0.548000; val_acc: 0.500000
(Iteration 4001 / 4900) loss: 1.501471
(Iteration 4101 / 4900) loss: 1.294143
(Iteration 4201 / 4900) loss: 1.653591
(Iteration 4301 / 4900) loss: 1.363827
(Iteration 4401 / 4900) loss: 1.245605
(Epoch 9 / 10) train acc: 0.554000; val_acc: 0.491000
```

```
(Iteration 4501 / 4900) loss: 1.421552
(Iteration 4601 / 4900) loss: 1.301349
(Iteration 4701 / 4900) loss: 1.281800
(Iteration 4801 / 4900) loss: 1.356831
(Epoch 10 / 10) train acc: 0.564000; val_acc: 0.508000
Best model updated! Current best_val_acc = 0.508
```





12 Test your model!

Run your best model on the validation and test sets. You should achieve above 48% accuracy on the validation set and the test set.

[16]:
```
y_val_pred = np.argmax(best_model.loss(data['X_val']), axis=1)
print('Validation set accuracy: ', (y_val_pred == data['y_val']).mean())
```

```
Validation set accuracy:  0.508
```

[17]:
```
y_test_pred = np.argmax(best_model.loss(data['X_test']), axis=1)
print('Test set accuracy: ', (y_test_pred == data['y_test']).mean())
```

```
Test set accuracy:  0.514
```

12.1 Inline Question 2:

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.

$\color{blue}{Your Answer:}$ 1, 3

$\color{blue}{Your Explanation:}$ Increasing the size of the training dataset can prevent the model from overfitting the data with restricted amount of training set. Also, increasing the regularization strength can help the model to generalize well.

features

August 6, 2021

[2]:
```
# This mounts your Google Drive to the Colab VM.
from google.colab import drive
drive.mount('/content/drive', force_remount=True)

# Enter the foldername in your Drive where you have saved the unzipped
# assignment folder, e.g. 'cs231n/assignments/assignment1/'
FOLDERNAME = 'CS231N/assignment1/'
assert FOLDERNAME is not None, "[!] Enter the foldername."

# Now that we've mounted your Drive, this ensures that
# the Python interpreter of the Colab VM can load
# python files from within it.
import sys
sys.path.append('/content/drive/My Drive/{}'.format(FOLDERNAME))

# This downloads the CIFAR-10 dataset to your Drive
# if it doesn't already exist.
%cd drive/My\ Drive/$FOLDERNAME/cs231n/datasets/
!bash get_datasets.sh
%cd /content/drive/My\ Drive/$FOLDERNAME
```

```
Mounted at /content/drive
/content/drive/My Drive/CS231N/assignment1/cs231n/datasets
/content/drive/My Drive/CS231N/assignment1
```

1 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 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.

[3]:
```
import random
import numpy as np
from cs231n.data_utils import load_CIFAR10
import matplotlib.pyplot as plt


%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-ipython
%load_ext autoreload
%autoreload 2
```

1.1 Load data

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

[4]:
```
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'

    # 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)

    # 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

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

1.2 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 own interest.

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.

[5]:
```
from cs231n.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.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
Done extracting features for 49000 / 49000 images
```

1.3 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.

[6]:
```python
# Use the validation set to tune the learning rate and regularization strength

from cs231n.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 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.       ␣
↪#
################################################################################
# *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****

for lr in learning_rates:
  for rs in regularization_strengths:
    svm = LinearSVM()
    svm.train(X_train_feats, y_train, learning_rate=lr, reg=rs, num_iters=1500,␣
↪verbose=False)

    y_val_pred = svm.predict(X_val_feats)
    val_accuracy = np.mean(y_val == y_val_pred)
    y_train_pred = svm.predict(X_train_feats)
    train_accuracy = np.mean(y_train == y_train_pred)
    results[(lr, rs)] = (train_accuracy, val_accuracy)

    if val_accuracy > best_val:
      best_val = val_accuracy
      best_svm = svm
```

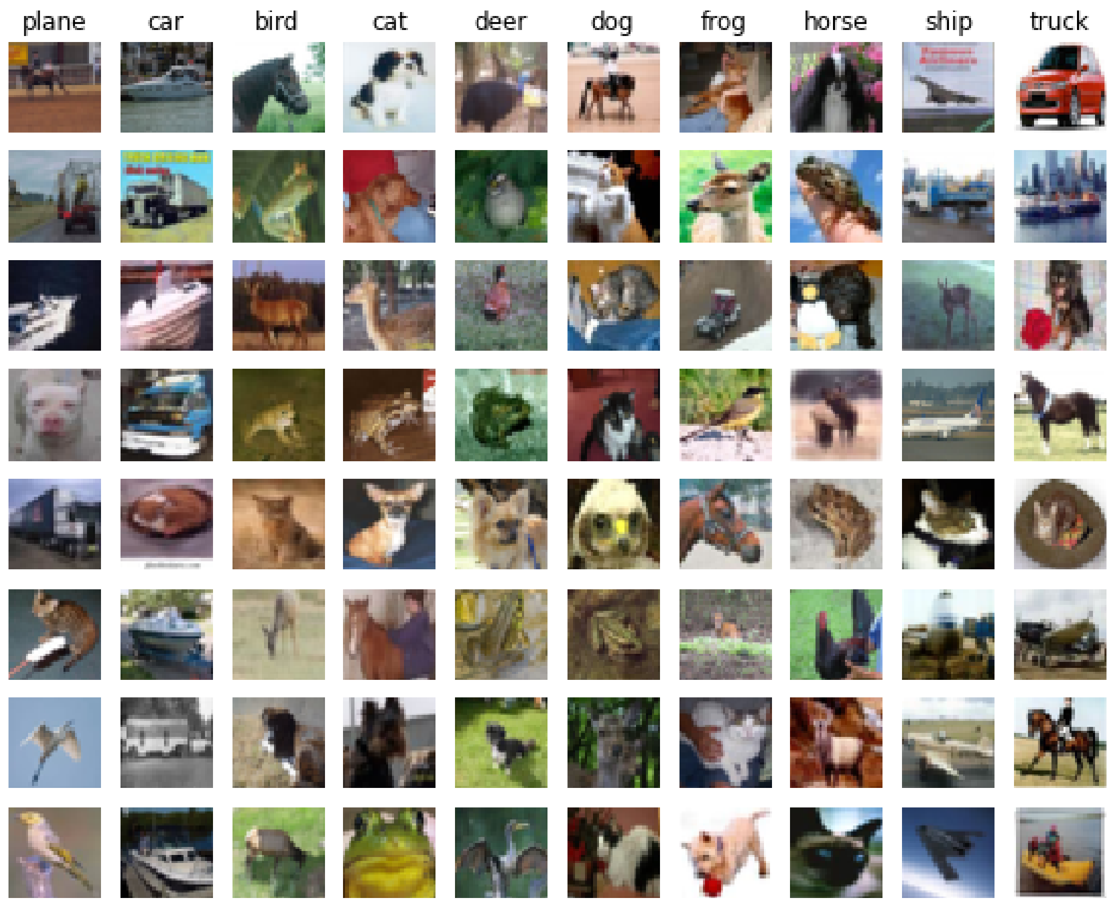
```
# *****END OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****

# 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: %f' % best_val)
```

```
lr 1.000000e-09 reg 5.000000e+04 train accuracy: 0.098531 val accuracy: 0.093000
lr 1.000000e-09 reg 5.000000e+05 train accuracy: 0.124245 val accuracy: 0.129000
lr 1.000000e-09 reg 5.000000e+06 train accuracy: 0.417367 val accuracy: 0.423000
lr 1.000000e-08 reg 5.000000e+04 train accuracy: 0.089265 val accuracy: 0.079000
lr 1.000000e-08 reg 5.000000e+05 train accuracy: 0.417327 val accuracy: 0.426000
lr 1.000000e-08 reg 5.000000e+06 train accuracy: 0.405204 val accuracy: 0.401000
lr 1.000000e-07 reg 5.000000e+04 train accuracy: 0.412653 val accuracy: 0.409000
lr 1.000000e-07 reg 5.000000e+05 train accuracy: 0.390224 val accuracy: 0.379000
lr 1.000000e-07 reg 5.000000e+06 train accuracy: 0.315367 val accuracy: 0.288000
best validation accuracy achieved: 0.426000
```

[7]:
```
# Evaluate your trained SVM on the test set: you should be able to get at least␣
 ↪0.40
y_test_pred = best_svm.predict(X_test_feats)
test_accuracy = np.mean(y_test == y_test_pred)
print(test_accuracy)
```

```
0.42
```

[8]:
```
# 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()
```



1.3.1 Inline question 1:

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

$Your Answer:$ For some, yes. Misclassified vehicles tend to be classified into different kind of vehicles. A similar tendency is observed in the images of animals.

1.4 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.

[9]:
```
# 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)
```

[24]:
```
from cs231n.classifiers.fc_net import TwoLayerNet
from cs231n.solver import Solver

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 want to   ␣
 ↪#
# cross-validate various parameters as in previous sections. Store your best  ␣
 ↪#
# model in the best_net variable.                                             ␣
 ↪#
################################################################################
# *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****

best_solver = None
num_trials = 30
input_dimension = X_train_feats.shape[1]


data = {}
data['X_train'] = X_train_feats
data['X_test'] = X_test_feats
data['X_val'] = X_val_feats
data['y_train'] = y_train
data['y_test'] = y_test
data['y_val'] = y_val


for i in range(num_trials):
```

```
_reg = np.random.uniform(0, 0.01)
_learning_rate = 10**(-np.random.uniform(1.1, 2.1))
_lr_decay = np.random.uniform(0.97, 1)
_num_epochs = 15
if _learning_rate < 10**-4:
  _num_epochs = 15

model = TwoLayerNet(input_dim = input_dimension, hidden_dim = 500, reg = _reg)
solver = Solver(model, data,
              update_rule='sgd',
              optim_config={
                'learning_rate': _learning_rate,
              },
              lr_decay=_lr_decay,
              num_epochs=1,
              batch_size=100,
              print_every=100)

print("Start computing trial {}/{}...".format(i, num_trials))
print("reg: {}, lr: {}, lr_decay: {}".format(_reg, _learning_rate, _lr_decay))

solver.train()
if solver.best_val_acc < 0.2:
  continue

model = TwoLayerNet(input_dim = input_dimension, hidden_dim = 500, reg = _reg)
solver = Solver(model, data,
              update_rule='sgd',
              optim_config={
                'learning_rate': _learning_rate,
              },
              lr_decay=_lr_decay,
              num_epochs=_num_epochs,
              batch_size=100,
              print_every=100)

print("Start computing trial {}/{}...".format(i, num_trials))
print("reg: {}, lr: {}, lr_decay: {}".format(_reg, _learning_rate, _lr_decay))

solver.train()

if best_net == None:
  best_net = model
  best_solver = solver

else:
    if solver.best_val_acc > best_solver.best_val_acc:
      best_net = model
      best_solver= solver
      print("Best model updated! Current best_val_acc = {}".format(best_solver.
 ↪best_val_acc))

    else:
      print("Best model not updated! Current best_val_acc = {}".
 ↪format(best_solver.best_val_acc))


# Run this cell to visualize training loss and train / val accuracy

plt.subplot(2, 1, 1)
plt.title('Training loss')
plt.plot(best_solver.loss_history, 'o')
plt.xlabel('Iteration')

plt.subplot(2, 1, 2)
plt.title('Accuracy')
plt.plot(best_solver.train_acc_history, '-o', label='train')
plt.plot(best_solver.val_acc_history, '-o', label='val')
plt.plot([0.5] * len(best_solver.val_acc_history), 'k--')
plt.xlabel('Epoch')
plt.legend(loc='lower right')
plt.gcf().set_size_inches(15, 12)
plt.show()

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

```
Start computing trial 0/30...
reg: 0.005355033185291348, lr: 0.010157568455885961, lr_decay:
0.9742423906226568
(Iteration 1 / 490) loss: 2.302806
(Epoch 0 / 1) train acc: 0.087000; val_acc: 0.102000
(Iteration 101 / 490) loss: 2.302460
(Iteration 201 / 490) loss: 2.302346
(Iteration 301 / 490) loss: 2.302635
(Iteration 401 / 490) loss: 2.300993
(Epoch 1 / 1) train acc: 0.167000; val_acc: 0.174000
Start computing trial 1/30...
reg: 0.009350176794328452, lr: 0.02427745170382816, lr_decay: 0.9998174611772865
(Iteration 1 / 490) loss: 2.302969
(Epoch 0 / 1) train acc: 0.092000; val_acc: 0.098000
(Iteration 101 / 490) loss: 2.302144
(Iteration 201 / 490) loss: 2.299344
(Iteration 301 / 490) loss: 2.286678
```

```
(Iteration 401 / 490) loss: 2.248632
(Epoch 1 / 1) train acc: 0.249000; val_acc: 0.259000
Start computing trial 1/30...
reg: 0.009350176794328452, lr: 0.02427745170382816, lr_decay: 0.9998174611772865
(Iteration 1 / 7350) loss: 2.302967
(Epoch 0 / 15) train acc: 0.071000; val_acc: 0.088000
(Iteration 101 / 7350) loss: 2.301809
(Iteration 201 / 7350) loss: 2.300822
(Iteration 301 / 7350) loss: 2.288749
(Iteration 401 / 7350) loss: 2.254520
(Epoch 1 / 15) train acc: 0.247000; val_acc: 0.269000
(Iteration 501 / 7350) loss: 2.149276
(Iteration 601 / 7350) loss: 1.987134
(Iteration 701 / 7350) loss: 2.047708
(Iteration 801 / 7350) loss: 1.903150
(Iteration 901 / 7350) loss: 1.777052
(Epoch 2 / 15) train acc: 0.377000; val_acc: 0.389000
(Iteration 1001 / 7350) loss: 1.768856
(Iteration 1101 / 7350) loss: 1.594457
(Iteration 1201 / 7350) loss: 1.572771
(Iteration 1301 / 7350) loss: 1.498551
(Iteration 1401 / 7350) loss: 1.547353
(Epoch 3 / 15) train acc: 0.448000; val_acc: 0.446000
(Iteration 1501 / 7350) loss: 1.532216
(Iteration 1601 / 7350) loss: 1.574016
(Iteration 1701 / 7350) loss: 1.632904
(Iteration 1801 / 7350) loss: 1.495156
(Iteration 1901 / 7350) loss: 1.543146
(Epoch 4 / 15) train acc: 0.495000; val_acc: 0.492000
(Iteration 2001 / 7350) loss: 1.339644
(Iteration 2101 / 7350) loss: 1.561899
(Iteration 2201 / 7350) loss: 1.471252
(Iteration 2301 / 7350) loss: 1.537934
(Iteration 2401 / 7350) loss: 1.531518
(Epoch 5 / 15) train acc: 0.519000; val_acc: 0.508000
(Iteration 2501 / 7350) loss: 1.467019
(Iteration 2601 / 7350) loss: 1.456103
(Iteration 2701 / 7350) loss: 1.556505
(Iteration 2801 / 7350) loss: 1.517764
(Iteration 2901 / 7350) loss: 1.290590
(Epoch 6 / 15) train acc: 0.544000; val_acc: 0.508000
(Iteration 3001 / 7350) loss: 1.440306
(Iteration 3101 / 7350) loss: 1.395166
(Iteration 3201 / 7350) loss: 1.414298
(Iteration 3301 / 7350) loss: 1.575744
(Iteration 3401 / 7350) loss: 1.384386
(Epoch 7 / 15) train acc: 0.507000; val_acc: 0.505000
(Iteration 3501 / 7350) loss: 1.339938
```

```
(Iteration 3601 / 7350) loss: 1.377855
(Iteration 3701 / 7350) loss: 1.444721
(Iteration 3801 / 7350) loss: 1.473617
(Iteration 3901 / 7350) loss: 1.485170
(Epoch 8 / 15) train acc: 0.534000; val_acc: 0.512000
(Iteration 4001 / 7350) loss: 1.444155
(Iteration 4101 / 7350) loss: 1.434082
(Iteration 4201 / 7350) loss: 1.452722
(Iteration 4301 / 7350) loss: 1.437395
(Iteration 4401 / 7350) loss: 1.507702
(Epoch 9 / 15) train acc: 0.515000; val_acc: 0.517000
(Iteration 4501 / 7350) loss: 1.392479
(Iteration 4601 / 7350) loss: 1.624383
(Iteration 4701 / 7350) loss: 1.497619
(Iteration 4801 / 7350) loss: 1.588297
(Epoch 10 / 15) train acc: 0.555000; val_acc: 0.509000
(Iteration 4901 / 7350) loss: 1.475712
(Iteration 5001 / 7350) loss: 1.274811
(Iteration 5101 / 7350) loss: 1.654669
(Iteration 5201 / 7350) loss: 1.573964
(Iteration 5301 / 7350) loss: 1.519324
(Epoch 11 / 15) train acc: 0.513000; val_acc: 0.512000
(Iteration 5401 / 7350) loss: 1.478945
(Iteration 5501 / 7350) loss: 1.469289
(Iteration 5601 / 7350) loss: 1.443708
(Iteration 5701 / 7350) loss: 1.391540
(Iteration 5801 / 7350) loss: 1.484967
(Epoch 12 / 15) train acc: 0.561000; val_acc: 0.519000
(Iteration 5901 / 7350) loss: 1.484855
(Iteration 6001 / 7350) loss: 1.593730
(Iteration 6101 / 7350) loss: 1.537169
(Iteration 6201 / 7350) loss: 1.512804
(Iteration 6301 / 7350) loss: 1.419505
(Epoch 13 / 15) train acc: 0.515000; val_acc: 0.518000
(Iteration 6401 / 7350) loss: 1.487885
(Iteration 6501 / 7350) loss: 1.443162
(Iteration 6601 / 7350) loss: 1.438391
(Iteration 6701 / 7350) loss: 1.513259
(Iteration 6801 / 7350) loss: 1.481000
(Epoch 14 / 15) train acc: 0.558000; val_acc: 0.532000
(Iteration 6901 / 7350) loss: 1.593111
(Iteration 7001 / 7350) loss: 1.382310
(Iteration 7101 / 7350) loss: 1.594772
(Iteration 7201 / 7350) loss: 1.243277
(Iteration 7301 / 7350) loss: 1.443853
(Epoch 15 / 15) train acc: 0.546000; val_acc: 0.526000
Start computing trial 2/30...
reg: 0.005218737594809696, lr: 0.03878662091419514, lr_decay: 0.988547358700182
```

```
(Iteration 1 / 490) loss: 2.302785
(Epoch 0 / 1) train acc: 0.097000; val_acc: 0.102000
(Iteration 101 / 490) loss: 2.300355
(Iteration 201 / 490) loss: 2.291449
(Iteration 301 / 490) loss: 2.212951
(Iteration 401 / 490) loss: 1.919501
(Epoch 1 / 1) train acc: 0.341000; val_acc: 0.331000
Start computing trial 2/30...
reg: 0.005218737594809696, lr: 0.03878662091419514, lr_decay: 0.988547358700182
(Iteration 1 / 7350) loss: 2.302787
(Epoch 0 / 15) train acc: 0.097000; val_acc: 0.094000
(Iteration 101 / 7350) loss: 2.300419
(Iteration 201 / 7350) loss: 2.288842
(Iteration 301 / 7350) loss: 2.135011
(Iteration 401 / 7350) loss: 2.019689
(Epoch 1 / 15) train acc: 0.359000; val_acc: 0.344000
(Iteration 501 / 7350) loss: 1.802063
(Iteration 601 / 7350) loss: 1.731131
(Iteration 701 / 7350) loss: 1.592434
(Iteration 801 / 7350) loss: 1.620647
(Iteration 901 / 7350) loss: 1.595960
(Epoch 2 / 15) train acc: 0.501000; val_acc: 0.463000
(Iteration 1001 / 7350) loss: 1.384101
(Iteration 1101 / 7350) loss: 1.538857
(Iteration 1201 / 7350) loss: 1.572153
(Iteration 1301 / 7350) loss: 1.426094
(Iteration 1401 / 7350) loss: 1.257196
(Epoch 3 / 15) train acc: 0.510000; val_acc: 0.498000
(Iteration 1501 / 7350) loss: 1.615232
(Iteration 1601 / 7350) loss: 1.475567
(Iteration 1701 / 7350) loss: 1.272728
(Iteration 1801 / 7350) loss: 1.404172
(Iteration 1901 / 7350) loss: 1.315265
(Epoch 4 / 15) train acc: 0.515000; val_acc: 0.507000
(Iteration 2001 / 7350) loss: 1.394369
(Iteration 2101 / 7350) loss: 1.342129
(Iteration 2201 / 7350) loss: 1.459410
(Iteration 2301 / 7350) loss: 1.571263
(Iteration 2401 / 7350) loss: 1.460122
(Epoch 5 / 15) train acc: 0.527000; val_acc: 0.516000
(Iteration 2501 / 7350) loss: 1.361771
(Iteration 2601 / 7350) loss: 1.536050
(Iteration 2701 / 7350) loss: 1.446327
(Iteration 2801 / 7350) loss: 1.419332
(Iteration 2901 / 7350) loss: 1.412444
(Epoch 6 / 15) train acc: 0.545000; val_acc: 0.525000
(Iteration 3001 / 7350) loss: 1.411997
(Iteration 3101 / 7350) loss: 1.547432
```

```
(Iteration 3201 / 7350) loss: 1.441380
(Iteration 3301 / 7350) loss: 1.488014
(Iteration 3401 / 7350) loss: 1.374820
(Epoch 7 / 15) train acc: 0.550000; val_acc: 0.526000
(Iteration 3501 / 7350) loss: 1.348443
(Iteration 3601 / 7350) loss: 1.535738
(Iteration 3701 / 7350) loss: 1.352167
(Iteration 3801 / 7350) loss: 1.325110
(Iteration 3901 / 7350) loss: 1.512603
(Epoch 8 / 15) train acc: 0.533000; val_acc: 0.531000
(Iteration 4001 / 7350) loss: 1.399803
(Iteration 4101 / 7350) loss: 1.566800
(Iteration 4201 / 7350) loss: 1.237713
(Iteration 4301 / 7350) loss: 1.421227
(Iteration 4401 / 7350) loss: 1.355693
(Epoch 9 / 15) train acc: 0.565000; val_acc: 0.538000
(Iteration 4501 / 7350) loss: 1.172991
(Iteration 4601 / 7350) loss: 1.426349
(Iteration 4701 / 7350) loss: 1.395721
(Iteration 4801 / 7350) loss: 1.390795
(Epoch 10 / 15) train acc: 0.530000; val_acc: 0.540000
(Iteration 4901 / 7350) loss: 1.298666
(Iteration 5001 / 7350) loss: 1.280748
(Iteration 5101 / 7350) loss: 1.244951
(Iteration 5201 / 7350) loss: 1.358496
(Iteration 5301 / 7350) loss: 1.432108
(Epoch 11 / 15) train acc: 0.551000; val_acc: 0.546000
(Iteration 5401 / 7350) loss: 1.159313
(Iteration 5501 / 7350) loss: 1.411029
(Iteration 5601 / 7350) loss: 1.416406
(Iteration 5701 / 7350) loss: 1.409006
(Iteration 5801 / 7350) loss: 1.316414
(Epoch 12 / 15) train acc: 0.528000; val_acc: 0.545000
(Iteration 5901 / 7350) loss: 1.218159
(Iteration 6001 / 7350) loss: 1.518259
(Iteration 6101 / 7350) loss: 1.409278
(Iteration 6201 / 7350) loss: 1.477680
(Iteration 6301 / 7350) loss: 1.544905
(Epoch 13 / 15) train acc: 0.557000; val_acc: 0.546000
(Iteration 6401 / 7350) loss: 1.258559
(Iteration 6501 / 7350) loss: 1.426830
(Iteration 6601 / 7350) loss: 1.438302
(Iteration 6701 / 7350) loss: 1.305362
(Iteration 6801 / 7350) loss: 1.408006
(Epoch 14 / 15) train acc: 0.600000; val_acc: 0.532000
(Iteration 6901 / 7350) loss: 1.535649
(Iteration 7001 / 7350) loss: 1.321982
(Iteration 7101 / 7350) loss: 1.521936
```

```
(Iteration 7201 / 7350) loss: 1.222715
(Iteration 7301 / 7350) loss: 1.309578
(Epoch 15 / 15) train acc: 0.586000; val_acc: 0.555000
Best model updated! Current best_val_acc = 0.555
Start computing trial 3/30...
reg: 0.007574700812870209, lr: 0.012822778845992908, lr_decay:
0.9728875251690785
(Iteration 1 / 490) loss: 2.302920
(Epoch 0 / 1) train acc: 0.092000; val_acc: 0.108000
(Iteration 101 / 490) loss: 2.302942
(Iteration 201 / 490) loss: 2.302279
(Iteration 301 / 490) loss: 2.301199
(Iteration 401 / 490) loss: 2.299053
(Epoch 1 / 1) train acc: 0.192000; val_acc: 0.203000
Start computing trial 3/30...
reg: 0.007574700812870209, lr: 0.012822778845992908, lr_decay:
0.9728875251690785
(Iteration 1 / 7350) loss: 2.302873
(Epoch 0 / 15) train acc: 0.112000; val_acc: 0.130000
(Iteration 101 / 7350) loss: 2.302324
(Iteration 201 / 7350) loss: 2.301864
(Iteration 301 / 7350) loss: 2.300903
(Iteration 401 / 7350) loss: 2.300933
(Epoch 1 / 15) train acc: 0.203000; val_acc: 0.188000
(Iteration 501 / 7350) loss: 2.294549
(Iteration 601 / 7350) loss: 2.290249
(Iteration 701 / 7350) loss: 2.264371
(Iteration 801 / 7350) loss: 2.225782
(Iteration 901 / 7350) loss: 2.167686
(Epoch 2 / 15) train acc: 0.260000; val_acc: 0.261000
(Iteration 1001 / 7350) loss: 2.081892
(Iteration 1101 / 7350) loss: 2.087351
(Iteration 1201 / 7350) loss: 2.030227
(Iteration 1301 / 7350) loss: 1.886224
(Iteration 1401 / 7350) loss: 1.959112
(Epoch 3 / 15) train acc: 0.324000; val_acc: 0.329000
(Iteration 1501 / 7350) loss: 1.846831
(Iteration 1601 / 7350) loss: 1.792557
(Iteration 1701 / 7350) loss: 1.733581
(Iteration 1801 / 7350) loss: 1.697625
(Iteration 1901 / 7350) loss: 1.700551
(Epoch 4 / 15) train acc: 0.399000; val_acc: 0.393000
(Iteration 2001 / 7350) loss: 1.620854
(Iteration 2101 / 7350) loss: 1.639673
(Iteration 2201 / 7350) loss: 1.719451
(Iteration 2301 / 7350) loss: 1.615564
(Iteration 2401 / 7350) loss: 1.595639
(Epoch 5 / 15) train acc: 0.444000; val_acc: 0.411000
```

```
(Iteration 2501 / 7350) loss: 1.618578
(Iteration 2601 / 7350) loss: 1.529880
(Iteration 2701 / 7350) loss: 1.490952
(Iteration 2801 / 7350) loss: 1.517031
(Iteration 2901 / 7350) loss: 1.434379
(Epoch 6 / 15) train acc: 0.445000; val_acc: 0.446000
(Iteration 3001 / 7350) loss: 1.506758
(Iteration 3101 / 7350) loss: 1.480802
(Iteration 3201 / 7350) loss: 1.612937
(Iteration 3301 / 7350) loss: 1.477004
(Iteration 3401 / 7350) loss: 1.508248
(Epoch 7 / 15) train acc: 0.451000; val_acc: 0.473000
(Iteration 3501 / 7350) loss: 1.549565
(Iteration 3601 / 7350) loss: 1.515728
(Iteration 3701 / 7350) loss: 1.371314
(Iteration 3801 / 7350) loss: 1.604810
(Iteration 3901 / 7350) loss: 1.528427
(Epoch 8 / 15) train acc: 0.498000; val_acc: 0.475000
(Iteration 4001 / 7350) loss: 1.644203
(Iteration 4101 / 7350) loss: 1.519099
(Iteration 4201 / 7350) loss: 1.548992
(Iteration 4301 / 7350) loss: 1.448106
(Iteration 4401 / 7350) loss: 1.603231
(Epoch 9 / 15) train acc: 0.511000; val_acc: 0.492000
(Iteration 4501 / 7350) loss: 1.529129
(Iteration 4601 / 7350) loss: 1.444393
(Iteration 4701 / 7350) loss: 1.549710
(Iteration 4801 / 7350) loss: 1.390855
(Epoch 10 / 15) train acc: 0.494000; val_acc: 0.496000
(Iteration 4901 / 7350) loss: 1.494489
(Iteration 5001 / 7350) loss: 1.456293
(Iteration 5101 / 7350) loss: 1.557653
(Iteration 5201 / 7350) loss: 1.472848
(Iteration 5301 / 7350) loss: 1.427442
(Epoch 11 / 15) train acc: 0.528000; val_acc: 0.501000
(Iteration 5401 / 7350) loss: 1.456167
(Iteration 5501 / 7350) loss: 1.507231
(Iteration 5601 / 7350) loss: 1.314225
(Iteration 5701 / 7350) loss: 1.622157
(Iteration 5801 / 7350) loss: 1.454655
(Epoch 12 / 15) train acc: 0.517000; val_acc: 0.502000
(Iteration 5901 / 7350) loss: 1.272455
(Iteration 6001 / 7350) loss: 1.461269
(Iteration 6101 / 7350) loss: 1.440081
(Iteration 6201 / 7350) loss: 1.293775
(Iteration 6301 / 7350) loss: 1.502353
(Epoch 13 / 15) train acc: 0.564000; val_acc: 0.505000
(Iteration 6401 / 7350) loss: 1.363236
```

```
(Iteration 6501 / 7350) loss: 1.446922
(Iteration 6601 / 7350) loss: 1.361238
(Iteration 6701 / 7350) loss: 1.487765
(Iteration 6801 / 7350) loss: 1.534703
(Epoch 14 / 15) train acc: 0.522000; val_acc: 0.511000
(Iteration 6901 / 7350) loss: 1.476298
(Iteration 7001 / 7350) loss: 1.555077
(Iteration 7101 / 7350) loss: 1.255151
(Iteration 7201 / 7350) loss: 1.369670
(Iteration 7301 / 7350) loss: 1.539417
(Epoch 15 / 15) train acc: 0.531000; val_acc: 0.512000
Best model not updated! Current best_val_acc = 0.555
Start computing trial 4/30...
reg: 0.007282616761299178, lr: 0.02925323367340108, lr_decay: 0.9971652271168309
(Iteration 1 / 490) loss: 2.302890
(Epoch 0 / 1) train acc: 0.103000; val_acc: 0.111000
(Iteration 101 / 490) loss: 2.301964
(Iteration 201 / 490) loss: 2.297838
(Iteration 301 / 490) loss: 2.260832
(Iteration 401 / 490) loss: 2.135942
(Epoch 1 / 1) train acc: 0.265000; val_acc: 0.268000
Start computing trial 4/30...
reg: 0.007282616761299178, lr: 0.02925323367340108, lr_decay: 0.9971652271168309
(Iteration 1 / 7350) loss: 2.302872
(Epoch 0 / 15) train acc: 0.111000; val_acc: 0.079000
(Iteration 101 / 7350) loss: 2.301890
(Iteration 201 / 7350) loss: 2.298243
(Iteration 301 / 7350) loss: 2.278719
(Iteration 401 / 7350) loss: 2.160605
(Epoch 1 / 15) train acc: 0.292000; val_acc: 0.286000
(Iteration 501 / 7350) loss: 1.996466
(Iteration 601 / 7350) loss: 1.885691
(Iteration 701 / 7350) loss: 1.784088
(Iteration 801 / 7350) loss: 1.684475
(Iteration 901 / 7350) loss: 1.631829
(Epoch 2 / 15) train acc: 0.422000; val_acc: 0.406000
(Iteration 1001 / 7350) loss: 1.713187
(Iteration 1101 / 7350) loss: 1.618458
(Iteration 1201 / 7350) loss: 1.549517
(Iteration 1301 / 7350) loss: 1.631900
(Iteration 1401 / 7350) loss: 1.438935
(Epoch 3 / 15) train acc: 0.481000; val_acc: 0.476000
(Iteration 1501 / 7350) loss: 1.425008
(Iteration 1601 / 7350) loss: 1.680936
(Iteration 1701 / 7350) loss: 1.495826
(Iteration 1801 / 7350) loss: 1.644203
(Iteration 1901 / 7350) loss: 1.566491
(Epoch 4 / 15) train acc: 0.522000; val_acc: 0.509000
```

```
(Iteration 2001 / 7350) loss: 1.453420
(Iteration 2101 / 7350) loss: 1.417468
(Iteration 2201 / 7350) loss: 1.331708
(Iteration 2301 / 7350) loss: 1.236321
(Iteration 2401 / 7350) loss: 1.433264
(Epoch 5 / 15) train acc: 0.507000; val_acc: 0.498000
(Iteration 2501 / 7350) loss: 1.542103
(Iteration 2601 / 7350) loss: 1.433481
(Iteration 2701 / 7350) loss: 1.493300
(Iteration 2801 / 7350) loss: 1.476914
(Iteration 2901 / 7350) loss: 1.336265
(Epoch 6 / 15) train acc: 0.529000; val_acc: 0.503000
(Iteration 3001 / 7350) loss: 1.365373
(Iteration 3101 / 7350) loss: 1.503945
(Iteration 3201 / 7350) loss: 1.523438
(Iteration 3301 / 7350) loss: 1.466447
(Iteration 3401 / 7350) loss: 1.492886
(Epoch 7 / 15) train acc: 0.520000; val_acc: 0.514000
(Iteration 3501 / 7350) loss: 1.512924
(Iteration 3601 / 7350) loss: 1.460958
(Iteration 3701 / 7350) loss: 1.606253
(Iteration 3801 / 7350) loss: 1.255947
(Iteration 3901 / 7350) loss: 1.503927
(Epoch 8 / 15) train acc: 0.558000; val_acc: 0.510000
(Iteration 4001 / 7350) loss: 1.445435
(Iteration 4101 / 7350) loss: 1.512339
(Iteration 4201 / 7350) loss: 1.325662
(Iteration 4301 / 7350) loss: 1.346217
(Iteration 4401 / 7350) loss: 1.452431
(Epoch 9 / 15) train acc: 0.546000; val_acc: 0.524000
(Iteration 4501 / 7350) loss: 1.558952
(Iteration 4601 / 7350) loss: 1.309077
(Iteration 4701 / 7350) loss: 1.328912
(Iteration 4801 / 7350) loss: 1.214073
(Epoch 10 / 15) train acc: 0.551000; val_acc: 0.528000
(Iteration 4901 / 7350) loss: 1.439896
(Iteration 5001 / 7350) loss: 1.382817
(Iteration 5101 / 7350) loss: 1.417945
(Iteration 5201 / 7350) loss: 1.324364
(Iteration 5301 / 7350) loss: 1.396731
(Epoch 11 / 15) train acc: 0.561000; val_acc: 0.542000
(Iteration 5401 / 7350) loss: 1.372716
(Iteration 5501 / 7350) loss: 1.336367
(Iteration 5601 / 7350) loss: 1.407801
(Iteration 5701 / 7350) loss: 1.445628
(Iteration 5801 / 7350) loss: 1.511756
(Epoch 12 / 15) train acc: 0.541000; val_acc: 0.532000
(Iteration 5901 / 7350) loss: 1.161436
```

```
(Iteration 6001 / 7350) loss: 1.411387
(Iteration 6101 / 7350) loss: 1.345551
(Iteration 6201 / 7350) loss: 1.346622
(Iteration 6301 / 7350) loss: 1.293693
(Epoch 13 / 15) train acc: 0.554000; val_acc: 0.550000
(Iteration 6401 / 7350) loss: 1.278371
(Iteration 6501 / 7350) loss: 1.502162
(Iteration 6601 / 7350) loss: 1.387626
(Iteration 6701 / 7350) loss: 1.298610
(Iteration 6801 / 7350) loss: 1.289669
(Epoch 14 / 15) train acc: 0.559000; val_acc: 0.540000
(Iteration 6901 / 7350) loss: 1.429391
(Iteration 7001 / 7350) loss: 1.345306
(Iteration 7101 / 7350) loss: 1.346396
(Iteration 7201 / 7350) loss: 1.242654
(Iteration 7301 / 7350) loss: 1.102959
(Epoch 15 / 15) train acc: 0.568000; val_acc: 0.547000
Best model not updated! Current best_val_acc = 0.555
Start computing trial 5/30...
reg: 0.00042143773994741O1, lr: 0.009204732661924307, lr_decay:
0.9706820792933177
(Iteration 1 / 490) loss: 2.302595
(Epoch 0 / 1) train acc: 0.093000; val_acc: 0.084000
(Iteration 101 / 490) loss: 2.302680
(Iteration 201 / 490) loss: 2.301998
(Iteration 301 / 490) loss: 2.302384
(Iteration 401 / 490) loss: 2.301236
(Epoch 1 / 1) train acc: 0.176000; val_acc: 0.188000
Start computing trial 6/30...
reg: 0.0033221877686387424, lr: 0.010903091173413107, lr_decay:
0.982446439834984
(Iteration 1 / 490) loss: 2.302762
(Epoch 0 / 1) train acc: 0.092000; val_acc: 0.084000
(Iteration 101 / 490) loss: 2.302593
(Iteration 201 / 490) loss: 2.302431
(Iteration 301 / 490) loss: 2.301445
(Iteration 401 / 490) loss: 2.300691
(Epoch 1 / 1) train acc: 0.234000; val_acc: 0.249000
Start computing trial 6/30...
reg: 0.0033221877686387424, lr: 0.010903091173413107, lr_decay:
0.982446439834984
(Iteration 1 / 7350) loss: 2.302712
(Epoch 0 / 15) train acc: 0.110000; val_acc: 0.129000
(Iteration 101 / 7350) loss: 2.302816
(Iteration 201 / 7350) loss: 2.301780
(Iteration 301 / 7350) loss: 2.301346
(Iteration 401 / 7350) loss: 2.300521
(Epoch 1 / 15) train acc: 0.216000; val_acc: 0.224000
```

```
(Iteration 501 / 7350) loss: 2.300250
(Iteration 601 / 7350) loss: 2.295623
(Iteration 701 / 7350) loss: 2.288516
(Iteration 801 / 7350) loss: 2.268871
(Iteration 901 / 7350) loss: 2.247625
(Epoch 2 / 15) train acc: 0.262000; val_acc: 0.255000
(Iteration 1001 / 7350) loss: 2.179614
(Iteration 1101 / 7350) loss: 2.167348
(Iteration 1201 / 7350) loss: 2.036104
(Iteration 1301 / 7350) loss: 2.006060
(Iteration 1401 / 7350) loss: 1.971138
(Epoch 3 / 15) train acc: 0.307000; val_acc: 0.300000
(Iteration 1501 / 7350) loss: 1.881135
(Iteration 1601 / 7350) loss: 1.915375
(Iteration 1701 / 7350) loss: 1.828411
(Iteration 1801 / 7350) loss: 1.869493
(Iteration 1901 / 7350) loss: 1.848387
(Epoch 4 / 15) train acc: 0.354000; val_acc: 0.379000
(Iteration 2001 / 7350) loss: 1.798242
(Iteration 2101 / 7350) loss: 1.783984
(Iteration 2201 / 7350) loss: 1.719881
(Iteration 2301 / 7350) loss: 1.649277
(Iteration 2401 / 7350) loss: 1.696806
(Epoch 5 / 15) train acc: 0.391000; val_acc: 0.413000
(Iteration 2501 / 7350) loss: 1.620483
(Iteration 2601 / 7350) loss: 1.651980
(Iteration 2701 / 7350) loss: 1.450502
(Iteration 2801 / 7350) loss: 1.426137
(Iteration 2901 / 7350) loss: 1.633495
(Epoch 6 / 15) train acc: 0.438000; val_acc: 0.436000
(Iteration 3001 / 7350) loss: 1.580910
(Iteration 3101 / 7350) loss: 1.523571
(Iteration 3201 / 7350) loss: 1.555591
(Iteration 3301 / 7350) loss: 1.693574
(Iteration 3401 / 7350) loss: 1.473112
(Epoch 7 / 15) train acc: 0.475000; val_acc: 0.456000
(Iteration 3501 / 7350) loss: 1.488965
(Iteration 3601 / 7350) loss: 1.572499
(Iteration 3701 / 7350) loss: 1.556306
(Iteration 3801 / 7350) loss: 1.518500
(Iteration 3901 / 7350) loss: 1.585555
(Epoch 8 / 15) train acc: 0.475000; val_acc: 0.490000
(Iteration 4001 / 7350) loss: 1.357644
(Iteration 4101 / 7350) loss: 1.663917
(Iteration 4201 / 7350) loss: 1.425760
(Iteration 4301 / 7350) loss: 1.480696
(Iteration 4401 / 7350) loss: 1.414038
(Epoch 9 / 15) train acc: 0.504000; val_acc: 0.485000
```

```
(Iteration 4501 / 7350) loss: 1.503819
(Iteration 4601 / 7350) loss: 1.535641
(Iteration 4701 / 7350) loss: 1.344089
(Iteration 4801 / 7350) loss: 1.395220
(Epoch 10 / 15) train acc: 0.524000; val_acc: 0.494000
(Iteration 4901 / 7350) loss: 1.308341
(Iteration 5001 / 7350) loss: 1.423912
(Iteration 5101 / 7350) loss: 1.637072
(Iteration 5201 / 7350) loss: 1.433637
(Iteration 5301 / 7350) loss: 1.285583
(Epoch 11 / 15) train acc: 0.509000; val_acc: 0.508000
(Iteration 5401 / 7350) loss: 1.598782
(Iteration 5501 / 7350) loss: 1.545904
(Iteration 5601 / 7350) loss: 1.479248
(Iteration 5701 / 7350) loss: 1.462011
(Iteration 5801 / 7350) loss: 1.624090
(Epoch 12 / 15) train acc: 0.544000; val_acc: 0.514000
(Iteration 5901 / 7350) loss: 1.417079
(Iteration 6001 / 7350) loss: 1.291312
(Iteration 6101 / 7350) loss: 1.480744
(Iteration 6201 / 7350) loss: 1.236403
(Iteration 6301 / 7350) loss: 1.269855
(Epoch 13 / 15) train acc: 0.488000; val_acc: 0.507000
(Iteration 6401 / 7350) loss: 1.337420
(Iteration 6501 / 7350) loss: 1.618666
(Iteration 6601 / 7350) loss: 1.413369
(Iteration 6701 / 7350) loss: 1.379020
(Iteration 6801 / 7350) loss: 1.355101
(Epoch 14 / 15) train acc: 0.540000; val_acc: 0.510000
(Iteration 6901 / 7350) loss: 1.496130
(Iteration 7001 / 7350) loss: 1.227695
(Iteration 7101 / 7350) loss: 1.258790
(Iteration 7201 / 7350) loss: 1.578756
(Iteration 7301 / 7350) loss: 1.440503
(Epoch 15 / 15) train acc: 0.546000; val_acc: 0.512000
Best model not updated! Current best_val_acc = 0.555
Start computing trial 7/30...
reg: 0.006253197562854539, lr: 0.053877298119779836, lr_decay:
0.9994275607783831
(Iteration 1 / 490) loss: 2.302880
(Epoch 0 / 1) train acc: 0.109000; val_acc: 0.118000
(Iteration 101 / 490) loss: 2.302480
(Iteration 201 / 490) loss: 2.236448
(Iteration 301 / 490) loss: 2.002137
(Iteration 401 / 490) loss: 1.758727
(Epoch 1 / 1) train acc: 0.392000; val_acc: 0.413000
Start computing trial 7/30...
reg: 0.006253197562854539, lr: 0.053877298119779836, lr_decay:
```

```
0.9994275607783831
(Iteration 1 / 7350) loss: 2.302826
(Epoch 0 / 15) train acc: 0.095000; val_acc: 0.079000
(Iteration 101 / 7350) loss: 2.302525
(Iteration 201 / 7350) loss: 2.217085
(Iteration 301 / 7350) loss: 1.973251
(Iteration 401 / 7350) loss: 1.719276
(Epoch 1 / 15) train acc: 0.408000; val_acc: 0.402000
(Iteration 501 / 7350) loss: 1.643334
(Iteration 601 / 7350) loss: 1.611990
(Iteration 701 / 7350) loss: 1.551016
(Iteration 801 / 7350) loss: 1.524799
(Iteration 901 / 7350) loss: 1.578818
(Epoch 2 / 15) train acc: 0.519000; val_acc: 0.489000
(Iteration 1001 / 7350) loss: 1.495130
(Iteration 1101 / 7350) loss: 1.381425
(Iteration 1201 / 7350) loss: 1.295193
(Iteration 1301 / 7350) loss: 1.434655
(Iteration 1401 / 7350) loss: 1.303959
(Epoch 3 / 15) train acc: 0.537000; val_acc: 0.499000
(Iteration 1501 / 7350) loss: 1.452523
(Iteration 1601 / 7350) loss: 1.279900
(Iteration 1701 / 7350) loss: 1.427422
(Iteration 1801 / 7350) loss: 1.412087
(Iteration 1901 / 7350) loss: 1.506599
(Epoch 4 / 15) train acc: 0.551000; val_acc: 0.516000
(Iteration 2001 / 7350) loss: 1.469259
(Iteration 2101 / 7350) loss: 1.534696
(Iteration 2201 / 7350) loss: 1.433774
(Iteration 2301 / 7350) loss: 1.401171
(Iteration 2401 / 7350) loss: 1.393458
(Epoch 5 / 15) train acc: 0.536000; val_acc: 0.528000
(Iteration 2501 / 7350) loss: 1.460554
(Iteration 2601 / 7350) loss: 1.425598
(Iteration 2701 / 7350) loss: 1.342778
(Iteration 2801 / 7350) loss: 1.311323
(Iteration 2901 / 7350) loss: 1.618359
(Epoch 6 / 15) train acc: 0.528000; val_acc: 0.536000
(Iteration 3001 / 7350) loss: 1.344049
(Iteration 3101 / 7350) loss: 1.518480
(Iteration 3201 / 7350) loss: 1.453873
(Iteration 3301 / 7350) loss: 1.398250
(Iteration 3401 / 7350) loss: 1.597901
(Epoch 7 / 15) train acc: 0.540000; val_acc: 0.536000
(Iteration 3501 / 7350) loss: 1.552533
(Iteration 3601 / 7350) loss: 1.505837
(Iteration 3701 / 7350) loss: 1.523993
(Iteration 3801 / 7350) loss: 1.228005
```

```
(Iteration 3901 / 7350) loss: 1.109899
(Epoch 8 / 15) train acc: 0.557000; val_acc: 0.533000
(Iteration 4001 / 7350) loss: 1.376538
(Iteration 4101 / 7350) loss: 1.319477
(Iteration 4201 / 7350) loss: 1.242625
(Iteration 4301 / 7350) loss: 1.339071
(Iteration 4401 / 7350) loss: 1.466355
(Epoch 9 / 15) train acc: 0.567000; val_acc: 0.549000
(Iteration 4501 / 7350) loss: 1.370956
(Iteration 4601 / 7350) loss: 1.507963
(Iteration 4701 / 7350) loss: 1.366040
(Iteration 4801 / 7350) loss: 1.415690
(Epoch 10 / 15) train acc: 0.576000; val_acc: 0.556000
(Iteration 4901 / 7350) loss: 1.224192
(Iteration 5001 / 7350) loss: 1.099104
(Iteration 5101 / 7350) loss: 1.181409
(Iteration 5201 / 7350) loss: 1.296024
(Iteration 5301 / 7350) loss: 1.344352
(Epoch 11 / 15) train acc: 0.589000; val_acc: 0.559000
(Iteration 5401 / 7350) loss: 1.410421
(Iteration 5501 / 7350) loss: 1.323451
(Iteration 5601 / 7350) loss: 1.347705
(Iteration 5701 / 7350) loss: 1.266472
(Iteration 5801 / 7350) loss: 1.334800
(Epoch 12 / 15) train acc: 0.577000; val_acc: 0.566000
(Iteration 5901 / 7350) loss: 1.267718
(Iteration 6001 / 7350) loss: 1.308260
(Iteration 6101 / 7350) loss: 1.280685
(Iteration 6201 / 7350) loss: 1.303411
(Iteration 6301 / 7350) loss: 1.247914
(Epoch 13 / 15) train acc: 0.578000; val_acc: 0.555000
(Iteration 6401 / 7350) loss: 1.444365
(Iteration 6501 / 7350) loss: 1.235529
(Iteration 6601 / 7350) loss: 1.387711
(Iteration 6701 / 7350) loss: 1.521557
(Iteration 6801 / 7350) loss: 1.126045
(Epoch 14 / 15) train acc: 0.602000; val_acc: 0.564000
(Iteration 6901 / 7350) loss: 1.334708
(Iteration 7001 / 7350) loss: 1.279997
(Iteration 7101 / 7350) loss: 1.387081
(Iteration 7201 / 7350) loss: 1.398945
(Iteration 7301 / 7350) loss: 1.178583
(Epoch 15 / 15) train acc: 0.593000; val_acc: 0.562000
Best model updated! Current best_val_acc = 0.566
Start computing trial 8/30...
reg: 0.00573474157823513, lr: 0.01970827608001487, lr_decay: 0.980476810587987
(Iteration 1 / 490) loss: 2.302818
(Epoch 0 / 1) train acc: 0.100000; val_acc: 0.097000
```

```
(Iteration 101 / 490) loss: 2.301948
(Iteration 201 / 490) loss: 2.300733
(Iteration 301 / 490) loss: 2.298307
(Iteration 401 / 490) loss: 2.287464
(Epoch 1 / 1) train acc: 0.251000; val_acc: 0.238000
Start computing trial 8/30...
reg: 0.00573474157823513, lr: 0.01970827608001487, lr_decay: 0.980476810587987
(Iteration 1 / 7350) loss: 2.302813
(Epoch 0 / 15) train acc: 0.096000; val_acc: 0.097000
(Iteration 101 / 7350) loss: 2.301203
(Iteration 201 / 7350) loss: 2.301927
(Iteration 301 / 7350) loss: 2.297007
(Iteration 401 / 7350) loss: 2.287442
(Epoch 1 / 15) train acc: 0.240000; val_acc: 0.225000
(Iteration 501 / 7350) loss: 2.239282
(Iteration 601 / 7350) loss: 2.207509
(Iteration 701 / 7350) loss: 2.107068
(Iteration 801 / 7350) loss: 1.939258
(Iteration 901 / 7350) loss: 1.873262
(Epoch 2 / 15) train acc: 0.354000; val_acc: 0.356000
(Iteration 1001 / 7350) loss: 1.658949
(Iteration 1101 / 7350) loss: 1.775926
(Iteration 1201 / 7350) loss: 1.687523
(Iteration 1301 / 7350) loss: 1.705253
(Iteration 1401 / 7350) loss: 1.717118
(Epoch 3 / 15) train acc: 0.451000; val_acc: 0.418000
(Iteration 1501 / 7350) loss: 1.597283
(Iteration 1601 / 7350) loss: 1.395960
(Iteration 1701 / 7350) loss: 1.456216
(Iteration 1801 / 7350) loss: 1.509317
(Iteration 1901 / 7350) loss: 1.569747
(Epoch 4 / 15) train acc: 0.477000; val_acc: 0.455000
(Iteration 2001 / 7350) loss: 1.556147
(Iteration 2101 / 7350) loss: 1.409279
(Iteration 2201 / 7350) loss: 1.512734
(Iteration 2301 / 7350) loss: 1.403138
(Iteration 2401 / 7350) loss: 1.548053
(Epoch 5 / 15) train acc: 0.523000; val_acc: 0.477000
(Iteration 2501 / 7350) loss: 1.493401
(Iteration 2601 / 7350) loss: 1.417094
(Iteration 2701 / 7350) loss: 1.511832
(Iteration 2801 / 7350) loss: 1.391907
(Iteration 2901 / 7350) loss: 1.568564
(Epoch 6 / 15) train acc: 0.512000; val_acc: 0.500000
(Iteration 3001 / 7350) loss: 1.365953
(Iteration 3101 / 7350) loss: 1.443874
(Iteration 3201 / 7350) loss: 1.278513
(Iteration 3301 / 7350) loss: 1.526485
```

```
(Iteration 3401 / 7350) loss: 1.222745
(Epoch 7 / 15) train acc: 0.531000; val_acc: 0.512000
(Iteration 3501 / 7350) loss: 1.505754
(Iteration 3601 / 7350) loss: 1.420901
(Iteration 3701 / 7350) loss: 1.338803
(Iteration 3801 / 7350) loss: 1.366702
(Iteration 3901 / 7350) loss: 1.384429
(Epoch 8 / 15) train acc: 0.516000; val_acc: 0.507000
(Iteration 4001 / 7350) loss: 1.211548
(Iteration 4101 / 7350) loss: 1.535441
(Iteration 4201 / 7350) loss: 1.365561
(Iteration 4301 / 7350) loss: 1.378305
(Iteration 4401 / 7350) loss: 1.493558
(Epoch 9 / 15) train acc: 0.530000; val_acc: 0.517000
(Iteration 4501 / 7350) loss: 1.433130
(Iteration 4601 / 7350) loss: 1.378153
(Iteration 4701 / 7350) loss: 1.234949
(Iteration 4801 / 7350) loss: 1.620114
(Epoch 10 / 15) train acc: 0.532000; val_acc: 0.513000
(Iteration 4901 / 7350) loss: 1.422337
(Iteration 5001 / 7350) loss: 1.409795
(Iteration 5101 / 7350) loss: 1.455774
(Iteration 5201 / 7350) loss: 1.494075
(Iteration 5301 / 7350) loss: 1.465804
(Epoch 11 / 15) train acc: 0.522000; val_acc: 0.511000
(Iteration 5401 / 7350) loss: 1.469828
(Iteration 5501 / 7350) loss: 1.519052
(Iteration 5601 / 7350) loss: 1.326468
(Iteration 5701 / 7350) loss: 1.402778
(Iteration 5801 / 7350) loss: 1.428338
(Epoch 12 / 15) train acc: 0.543000; val_acc: 0.511000
(Iteration 5901 / 7350) loss: 1.447433
(Iteration 6001 / 7350) loss: 1.465814
(Iteration 6101 / 7350) loss: 1.297175
(Iteration 6201 / 7350) loss: 1.498054
(Iteration 6301 / 7350) loss: 1.296893
(Epoch 13 / 15) train acc: 0.549000; val_acc: 0.514000
(Iteration 6401 / 7350) loss: 1.377375
(Iteration 6501 / 7350) loss: 1.327941
(Iteration 6601 / 7350) loss: 1.422640
(Iteration 6701 / 7350) loss: 1.368007
(Iteration 6801 / 7350) loss: 1.185931
(Epoch 14 / 15) train acc: 0.566000; val_acc: 0.514000
(Iteration 6901 / 7350) loss: 1.447920
(Iteration 7001 / 7350) loss: 1.380285
(Iteration 7101 / 7350) loss: 1.383812
(Iteration 7201 / 7350) loss: 1.323443
(Iteration 7301 / 7350) loss: 1.382808
```

```
(Epoch 15 / 15) train acc: 0.563000; val_acc: 0.510000
Best model not updated! Current best_val_acc = 0.566
Start computing trial 9/30...
reg: 0.008388726961207635, lr: 0.012224173606691197, lr_decay:
0.9722168494904843
(Iteration 1 / 490) loss: 2.302932
(Epoch 0 / 1) train acc: 0.073000; val_acc: 0.087000
(Iteration 101 / 490) loss: 2.303025
(Iteration 201 / 490) loss: 2.302669
(Iteration 301 / 490) loss: 2.301223
(Iteration 401 / 490) loss: 2.299200
(Epoch 1 / 1) train acc: 0.162000; val_acc: 0.158000
Start computing trial 10/30...
reg: 0.00648675462566049, lr: 0.009333057235119756, lr_decay: 0.9845320122870519
(Iteration 1 / 490) loss: 2.302848
(Epoch 0 / 1) train acc: 0.074000; val_acc: 0.074000
(Iteration 101 / 490) loss: 2.302727
(Iteration 201 / 490) loss: 2.302292
(Iteration 301 / 490) loss: 2.301935
(Iteration 401 / 490) loss: 2.301086
(Epoch 1 / 1) train acc: 0.146000; val_acc: 0.111000
Start computing trial 11/30...
reg: 0.001505782455784338, lr: 0.07419335086075014, lr_decay: 0.9835593949465361
(Iteration 1 / 490) loss: 2.302620
(Epoch 0 / 1) train acc: 0.113000; val_acc: 0.078000
(Iteration 101 / 490) loss: 2.288075
(Iteration 201 / 490) loss: 1.915568
(Iteration 301 / 490) loss: 1.910521
(Iteration 401 / 490) loss: 1.529970
(Epoch 1 / 1) train acc: 0.469000; val_acc: 0.478000
Start computing trial 11/30...
reg: 0.001505782455784338, lr: 0.07419335086075014, lr_decay: 0.9835593949465361
(Iteration 1 / 7350) loss: 2.302644
(Epoch 0 / 15) train acc: 0.106000; val_acc: 0.103000
(Iteration 101 / 7350) loss: 2.288871
(Iteration 201 / 7350) loss: 1.949944
(Iteration 301 / 7350) loss: 1.748785
(Iteration 401 / 7350) loss: 1.617184
(Epoch 1 / 15) train acc: 0.454000; val_acc: 0.449000
(Iteration 501 / 7350) loss: 1.562443
(Iteration 601 / 7350) loss: 1.474231
(Iteration 701 / 7350) loss: 1.300315
(Iteration 801 / 7350) loss: 1.460093
(Iteration 901 / 7350) loss: 1.307388
(Epoch 2 / 15) train acc: 0.498000; val_acc: 0.518000
(Iteration 1001 / 7350) loss: 1.438669
(Iteration 1101 / 7350) loss: 1.465479
(Iteration 1201 / 7350) loss: 1.420876
```

```
(Iteration 1301 / 7350) loss: 1.422256
(Iteration 1401 / 7350) loss: 1.424496
(Epoch 3 / 15) train acc: 0.518000; val_acc: 0.527000
(Iteration 1501 / 7350) loss: 1.106800
(Iteration 1601 / 7350) loss: 1.409655
(Iteration 1701 / 7350) loss: 1.299791
(Iteration 1801 / 7350) loss: 1.387699
(Iteration 1901 / 7350) loss: 1.324044
(Epoch 4 / 15) train acc: 0.530000; val_acc: 0.531000
(Iteration 2001 / 7350) loss: 1.249239
(Iteration 2101 / 7350) loss: 1.314157
(Iteration 2201 / 7350) loss: 1.123104
(Iteration 2301 / 7350) loss: 1.424938
(Iteration 2401 / 7350) loss: 1.360312
(Epoch 5 / 15) train acc: 0.559000; val_acc: 0.547000
(Iteration 2501 / 7350) loss: 1.123656
(Iteration 2601 / 7350) loss: 1.290979
(Iteration 2701 / 7350) loss: 1.148031
(Iteration 2801 / 7350) loss: 1.135541
(Iteration 2901 / 7350) loss: 1.263686
(Epoch 6 / 15) train acc: 0.593000; val_acc: 0.563000
(Iteration 3001 / 7350) loss: 1.408493
(Iteration 3101 / 7350) loss: 1.342517
(Iteration 3201 / 7350) loss: 1.280735
(Iteration 3301 / 7350) loss: 1.233004
(Iteration 3401 / 7350) loss: 1.125422
(Epoch 7 / 15) train acc: 0.612000; val_acc: 0.576000
(Iteration 3501 / 7350) loss: 1.206242
(Iteration 3601 / 7350) loss: 1.438082
(Iteration 3701 / 7350) loss: 1.390050
(Iteration 3801 / 7350) loss: 1.240604
(Iteration 3901 / 7350) loss: 1.226438
(Epoch 8 / 15) train acc: 0.633000; val_acc: 0.580000
(Iteration 4001 / 7350) loss: 1.243097
(Iteration 4101 / 7350) loss: 1.142489
(Iteration 4201 / 7350) loss: 1.358396
(Iteration 4301 / 7350) loss: 1.333127
(Iteration 4401 / 7350) loss: 1.272890
(Epoch 9 / 15) train acc: 0.597000; val_acc: 0.572000
(Iteration 4501 / 7350) loss: 0.999199
(Iteration 4601 / 7350) loss: 1.025531
(Iteration 4701 / 7350) loss: 1.107367
(Iteration 4801 / 7350) loss: 1.214456
(Epoch 10 / 15) train acc: 0.645000; val_acc: 0.589000
(Iteration 4901 / 7350) loss: 1.176137
(Iteration 5001 / 7350) loss: 1.156451
(Iteration 5101 / 7350) loss: 1.159466
(Iteration 5201 / 7350) loss: 1.130413
```

```
(Iteration 5301 / 7350) loss: 1.041799
(Epoch 11 / 15) train acc: 0.645000; val_acc: 0.585000
(Iteration 5401 / 7350) loss: 0.981067
(Iteration 5501 / 7350) loss: 1.129705
(Iteration 5601 / 7350) loss: 1.065888
(Iteration 5701 / 7350) loss: 1.017494
(Iteration 5801 / 7350) loss: 1.111578
(Epoch 12 / 15) train acc: 0.665000; val_acc: 0.591000
(Iteration 5901 / 7350) loss: 1.116588
(Iteration 6001 / 7350) loss: 1.047192
(Iteration 6101 / 7350) loss: 1.111793
(Iteration 6201 / 7350) loss: 1.170900
(Iteration 6301 / 7350) loss: 1.097336
(Epoch 13 / 15) train acc: 0.628000; val_acc: 0.585000
(Iteration 6401 / 7350) loss: 1.165719
(Iteration 6501 / 7350) loss: 1.198025
(Iteration 6601 / 7350) loss: 1.087751
(Iteration 6701 / 7350) loss: 0.981192
(Iteration 6801 / 7350) loss: 1.030727
(Epoch 14 / 15) train acc: 0.663000; val_acc: 0.606000
(Iteration 6901 / 7350) loss: 1.048858
(Iteration 7001 / 7350) loss: 1.140336
(Iteration 7101 / 7350) loss: 0.995084
(Iteration 7201 / 7350) loss: 1.151980
(Iteration 7301 / 7350) loss: 1.083983
(Epoch 15 / 15) train acc: 0.677000; val_acc: 0.603000
Best model updated! Current best_val_acc = 0.606
Start computing trial 12/30...
reg: 0.000500442632878606, lr: 0.0164719957140072, lr_decay: 0.9857650636543086
(Iteration 1 / 490) loss: 2.302605
(Epoch 0 / 1) train acc: 0.091000; val_acc: 0.111000
(Iteration 101 / 490) loss: 2.301995
(Iteration 201 / 490) loss: 2.301826
(Iteration 301 / 490) loss: 2.300669
(Iteration 401 / 490) loss: 2.295438
(Epoch 1 / 1) train acc: 0.294000; val_acc: 0.283000
Start computing trial 12/30...
reg: 0.000500442632878606, lr: 0.0164719957140072, lr_decay: 0.9857650636543086
(Iteration 1 / 7350) loss: 2.302587
(Epoch 0 / 15) train acc: 0.105000; val_acc: 0.113000
(Iteration 101 / 7350) loss: 2.301823
(Iteration 201 / 7350) loss: 2.301536
(Iteration 301 / 7350) loss: 2.297908
(Iteration 401 / 7350) loss: 2.295187
(Epoch 1 / 15) train acc: 0.253000; val_acc: 0.219000
(Iteration 501 / 7350) loss: 2.269021
(Iteration 601 / 7350) loss: 2.246882
(Iteration 701 / 7350) loss: 2.120524
```

```
(Iteration 801 / 7350) loss: 2.099304
(Iteration 901 / 7350) loss: 1.928556
(Epoch 2 / 15) train acc: 0.292000; val_acc: 0.309000
(Iteration 1001 / 7350) loss: 1.791173
(Iteration 1101 / 7350) loss: 1.943023
(Iteration 1201 / 7350) loss: 1.773139
(Iteration 1301 / 7350) loss: 1.866999
(Iteration 1401 / 7350) loss: 1.642222
(Epoch 3 / 15) train acc: 0.401000; val_acc: 0.395000
(Iteration 1501 / 7350) loss: 1.520341
(Iteration 1601 / 7350) loss: 1.558185
(Iteration 1701 / 7350) loss: 1.610674
(Iteration 1801 / 7350) loss: 1.593506
(Iteration 1901 / 7350) loss: 1.430980
(Epoch 4 / 15) train acc: 0.484000; val_acc: 0.446000
(Iteration 2001 / 7350) loss: 1.525666
(Iteration 2101 / 7350) loss: 1.666063
(Iteration 2201 / 7350) loss: 1.482613
(Iteration 2301 / 7350) loss: 1.522265
(Iteration 2401 / 7350) loss: 1.475370
(Epoch 5 / 15) train acc: 0.476000; val_acc: 0.482000
(Iteration 2501 / 7350) loss: 1.608928
(Iteration 2601 / 7350) loss: 1.427413
(Iteration 2701 / 7350) loss: 1.350115
(Iteration 2801 / 7350) loss: 1.406488
(Iteration 2901 / 7350) loss: 1.458089
(Epoch 6 / 15) train acc: 0.506000; val_acc: 0.498000
(Iteration 3001 / 7350) loss: 1.483113
(Iteration 3101 / 7350) loss: 1.649403
(Iteration 3201 / 7350) loss: 1.434912
(Iteration 3301 / 7350) loss: 1.417332
(Iteration 3401 / 7350) loss: 1.639717
(Epoch 7 / 15) train acc: 0.505000; val_acc: 0.504000
(Iteration 3501 / 7350) loss: 1.454811
(Iteration 3601 / 7350) loss: 1.226488
(Iteration 3701 / 7350) loss: 1.247579
(Iteration 3801 / 7350) loss: 1.409565
(Iteration 3901 / 7350) loss: 1.334919
(Epoch 8 / 15) train acc: 0.517000; val_acc: 0.512000
(Iteration 4001 / 7350) loss: 1.422672
(Iteration 4101 / 7350) loss: 1.395061
(Iteration 4201 / 7350) loss: 1.401410
(Iteration 4301 / 7350) loss: 1.298189
(Iteration 4401 / 7350) loss: 1.512494
(Epoch 9 / 15) train acc: 0.522000; val_acc: 0.510000
(Iteration 4501 / 7350) loss: 1.246139
(Iteration 4601 / 7350) loss: 1.232862
(Iteration 4701 / 7350) loss: 1.350279
```

```
(Iteration 4801 / 7350) loss: 1.306529
(Epoch 10 / 15) train acc: 0.535000; val_acc: 0.510000
(Iteration 4901 / 7350) loss: 1.259539
(Iteration 5001 / 7350) loss: 1.368500
(Iteration 5101 / 7350) loss: 1.318356
(Iteration 5201 / 7350) loss: 1.461882
(Iteration 5301 / 7350) loss: 1.320381
(Epoch 11 / 15) train acc: 0.537000; val_acc: 0.514000
(Iteration 5401 / 7350) loss: 1.327985
(Iteration 5501 / 7350) loss: 1.319343
(Iteration 5601 / 7350) loss: 1.330232
(Iteration 5701 / 7350) loss: 1.238451
(Iteration 5801 / 7350) loss: 1.251181
(Epoch 12 / 15) train acc: 0.544000; val_acc: 0.514000
(Iteration 5901 / 7350) loss: 1.349453
(Iteration 6001 / 7350) loss: 1.304226
(Iteration 6101 / 7350) loss: 1.274528
(Iteration 6201 / 7350) loss: 1.453862
(Iteration 6301 / 7350) loss: 1.470147
(Epoch 13 / 15) train acc: 0.533000; val_acc: 0.523000
(Iteration 6401 / 7350) loss: 1.349094
(Iteration 6501 / 7350) loss: 1.397161
(Iteration 6601 / 7350) loss: 1.207534
(Iteration 6701 / 7350) loss: 1.342458
(Iteration 6801 / 7350) loss: 1.330104
(Epoch 14 / 15) train acc: 0.531000; val_acc: 0.528000
(Iteration 6901 / 7350) loss: 1.282393
(Iteration 7001 / 7350) loss: 1.319364
(Iteration 7101 / 7350) loss: 1.233648
(Iteration 7201 / 7350) loss: 1.275915
(Iteration 7301 / 7350) loss: 1.313340
(Epoch 15 / 15) train acc: 0.543000; val_acc: 0.532000
Best model not updated! Current best_val_acc = 0.606
Start computing trial 13/30...
reg: 0.0034937304133994184, lr: 0.00840054753046216, lr_decay:
0.9997315272292893
(Iteration 1 / 490) loss: 2.302760
(Epoch 0 / 1) train acc: 0.089000; val_acc: 0.077000
(Iteration 101 / 490) loss: 2.302472
(Iteration 201 / 490) loss: 2.302140
(Iteration 301 / 490) loss: 2.301831
(Iteration 401 / 490) loss: 2.302515
(Epoch 1 / 1) train acc: 0.108000; val_acc: 0.078000
Start computing trial 14/30...
reg: 0.005269992791909177, lr: 0.0179740575036515, lr_decay: 0.9700837038162321
(Iteration 1 / 490) loss: 2.302752
(Epoch 0 / 1) train acc: 0.097000; val_acc: 0.079000
(Iteration 101 / 490) loss: 2.301941
```

```
(Iteration 201 / 490) loss: 2.301549
(Iteration 301 / 490) loss: 2.297962
(Iteration 401 / 490) loss: 2.295967
(Epoch 1 / 1) train acc: 0.228000; val_acc: 0.194000
Start computing trial 15/30...
reg: 0.005570204245558476, lr: 0.05609838726868371, lr_decay: 0.9725303275037008
(Iteration 1 / 490) loss: 2.302829
(Epoch 0 / 1) train acc: 0.105000; val_acc: 0.098000
(Iteration 101 / 490) loss: 2.299440
(Iteration 201 / 490) loss: 2.190400
(Iteration 301 / 490) loss: 1.979636
(Iteration 401 / 490) loss: 1.854319
(Epoch 1 / 1) train acc: 0.406000; val_acc: 0.402000
Start computing trial 15/30...
reg: 0.005570204245558476, lr: 0.05609838726868371, lr_decay: 0.9725303275037008
(Iteration 1 / 7350) loss: 2.302859
(Epoch 0 / 15) train acc: 0.123000; val_acc: 0.106000
(Iteration 101 / 7350) loss: 2.300275
(Iteration 201 / 7350) loss: 2.240662
(Iteration 301 / 7350) loss: 1.954783
(Iteration 401 / 7350) loss: 1.730228
(Epoch 1 / 15) train acc: 0.440000; val_acc: 0.400000
(Iteration 501 / 7350) loss: 1.640339
(Iteration 601 / 7350) loss: 1.443279
(Iteration 701 / 7350) loss: 1.504304
(Iteration 801 / 7350) loss: 1.633730
(Iteration 901 / 7350) loss: 1.355460
(Epoch 2 / 15) train acc: 0.524000; val_acc: 0.490000
(Iteration 1001 / 7350) loss: 1.644923
(Iteration 1101 / 7350) loss: 1.481605
(Iteration 1201 / 7350) loss: 1.428290
(Iteration 1301 / 7350) loss: 1.332099
(Iteration 1401 / 7350) loss: 1.255380
(Epoch 3 / 15) train acc: 0.522000; val_acc: 0.504000
(Iteration 1501 / 7350) loss: 1.344439
(Iteration 1601 / 7350) loss: 1.561539
(Iteration 1701 / 7350) loss: 1.507882
(Iteration 1801 / 7350) loss: 1.366086
(Iteration 1901 / 7350) loss: 1.312811
(Epoch 4 / 15) train acc: 0.551000; val_acc: 0.513000
(Iteration 2001 / 7350) loss: 1.245450
(Iteration 2101 / 7350) loss: 1.469337
(Iteration 2201 / 7350) loss: 1.445129
(Iteration 2301 / 7350) loss: 1.405330
(Iteration 2401 / 7350) loss: 1.424229
(Epoch 5 / 15) train acc: 0.541000; val_acc: 0.530000
(Iteration 2501 / 7350) loss: 1.386357
(Iteration 2601 / 7350) loss: 1.372334
```

```
(Iteration 2701 / 7350) loss: 1.410259
(Iteration 2801 / 7350) loss: 1.385650
(Iteration 2901 / 7350) loss: 1.467899
(Epoch 6 / 15) train acc: 0.526000; val_acc: 0.530000
(Iteration 3001 / 7350) loss: 1.375712
(Iteration 3101 / 7350) loss: 1.402359
(Iteration 3201 / 7350) loss: 1.184134
(Iteration 3301 / 7350) loss: 1.288805
(Iteration 3401 / 7350) loss: 1.625844
(Epoch 7 / 15) train acc: 0.556000; val_acc: 0.553000
(Iteration 3501 / 7350) loss: 1.387651
(Iteration 3601 / 7350) loss: 1.276834
(Iteration 3701 / 7350) loss: 1.474652
(Iteration 3801 / 7350) loss: 1.284481
(Iteration 3901 / 7350) loss: 1.329975
(Epoch 8 / 15) train acc: 0.589000; val_acc: 0.543000
(Iteration 4001 / 7350) loss: 1.282231
(Iteration 4101 / 7350) loss: 1.266336
(Iteration 4201 / 7350) loss: 1.235220
(Iteration 4301 / 7350) loss: 1.528381
(Iteration 4401 / 7350) loss: 1.412852
(Epoch 9 / 15) train acc: 0.574000; val_acc: 0.544000
(Iteration 4501 / 7350) loss: 1.299221
(Iteration 4601 / 7350) loss: 1.325377
(Iteration 4701 / 7350) loss: 1.293451
(Iteration 4801 / 7350) loss: 1.294160
(Epoch 10 / 15) train acc: 0.573000; val_acc: 0.546000
(Iteration 4901 / 7350) loss: 1.394086
(Iteration 5001 / 7350) loss: 1.585986
(Iteration 5101 / 7350) loss: 1.248478
(Iteration 5201 / 7350) loss: 1.299369
(Iteration 5301 / 7350) loss: 1.344957
(Epoch 11 / 15) train acc: 0.575000; val_acc: 0.559000
(Iteration 5401 / 7350) loss: 1.487419
(Iteration 5501 / 7350) loss: 1.393244
(Iteration 5601 / 7350) loss: 1.339870
(Iteration 5701 / 7350) loss: 1.082167
(Iteration 5801 / 7350) loss: 1.347653
(Epoch 12 / 15) train acc: 0.569000; val_acc: 0.553000
(Iteration 5901 / 7350) loss: 1.404344
(Iteration 6001 / 7350) loss: 1.194965
(Iteration 6101 / 7350) loss: 1.312530
(Iteration 6201 / 7350) loss: 1.236593
(Iteration 6301 / 7350) loss: 1.338464
(Epoch 13 / 15) train acc: 0.592000; val_acc: 0.558000
(Iteration 6401 / 7350) loss: 1.334042
(Iteration 6501 / 7350) loss: 1.318961
(Iteration 6601 / 7350) loss: 1.293340
```

```
(Iteration 6701 / 7350) loss: 1.342059
(Iteration 6801 / 7350) loss: 1.419586
(Epoch 14 / 15) train acc: 0.596000; val_acc: 0.557000
(Iteration 6901 / 7350) loss: 1.234435
(Iteration 7001 / 7350) loss: 1.297426
(Iteration 7101 / 7350) loss: 1.442916
(Iteration 7201 / 7350) loss: 1.304303
(Iteration 7301 / 7350) loss: 1.319209
(Epoch 15 / 15) train acc: 0.624000; val_acc: 0.567000
Best model not updated! Current best_val_acc = 0.606
Start computing trial 16/30...
reg: 0.004280014420159561, lr: 0.03741959235635439, lr_decay: 0.9728148387665511
(Iteration 1 / 490) loss: 2.302764
(Epoch 0 / 1) train acc: 0.102000; val_acc: 0.107000
(Iteration 101 / 490) loss: 2.303025
(Iteration 201 / 490) loss: 2.291554
(Iteration 301 / 490) loss: 2.171964
(Iteration 401 / 490) loss: 1.993966
(Epoch 1 / 1) train acc: 0.334000; val_acc: 0.331000
Start computing trial 16/30...
reg: 0.004280014420159561, lr: 0.03741959235635439, lr_decay: 0.9728148387665511
(Iteration 1 / 7350) loss: 2.302753
(Epoch 0 / 15) train acc: 0.109000; val_acc: 0.102000
(Iteration 101 / 7350) loss: 2.300966
(Iteration 201 / 7350) loss: 2.292386
(Iteration 301 / 7350) loss: 2.159389
(Iteration 401 / 7350) loss: 1.992290
(Epoch 1 / 15) train acc: 0.323000; val_acc: 0.344000
(Iteration 501 / 7350) loss: 1.805362
(Iteration 601 / 7350) loss: 1.741705
(Iteration 701 / 7350) loss: 1.681676
(Iteration 801 / 7350) loss: 1.506262
(Iteration 901 / 7350) loss: 1.485539
(Epoch 2 / 15) train acc: 0.477000; val_acc: 0.467000
(Iteration 1001 / 7350) loss: 1.431442
(Iteration 1101 / 7350) loss: 1.447488
(Iteration 1201 / 7350) loss: 1.559266
(Iteration 1301 / 7350) loss: 1.383683
(Iteration 1401 / 7350) loss: 1.458382
(Epoch 3 / 15) train acc: 0.496000; val_acc: 0.504000
(Iteration 1501 / 7350) loss: 1.467866
(Iteration 1601 / 7350) loss: 1.516645
(Iteration 1701 / 7350) loss: 1.463337
(Iteration 1801 / 7350) loss: 1.402357
(Iteration 1901 / 7350) loss: 1.293124
(Epoch 4 / 15) train acc: 0.530000; val_acc: 0.508000
(Iteration 2001 / 7350) loss: 1.265087
(Iteration 2101 / 7350) loss: 1.474255
```

```
(Iteration 2201 / 7350) loss: 1.389195
(Iteration 2301 / 7350) loss: 1.317954
(Iteration 2401 / 7350) loss: 1.450072
(Epoch 5 / 15) train acc: 0.510000; val_acc: 0.510000
(Iteration 2501 / 7350) loss: 1.334799
(Iteration 2601 / 7350) loss: 1.310418
(Iteration 2701 / 7350) loss: 1.333535
(Iteration 2801 / 7350) loss: 1.312398
(Iteration 2901 / 7350) loss: 1.409372
(Epoch 6 / 15) train acc: 0.532000; val_acc: 0.523000
(Iteration 3001 / 7350) loss: 1.290906
(Iteration 3101 / 7350) loss: 1.241789
(Iteration 3201 / 7350) loss: 1.429463
(Iteration 3301 / 7350) loss: 1.378919
(Iteration 3401 / 7350) loss: 1.300893
(Epoch 7 / 15) train acc: 0.554000; val_acc: 0.519000
(Iteration 3501 / 7350) loss: 1.198722
(Iteration 3601 / 7350) loss: 1.450688
(Iteration 3701 / 7350) loss: 1.291707
(Iteration 3801 / 7350) loss: 1.417193
(Iteration 3901 / 7350) loss: 1.381543
(Epoch 8 / 15) train acc: 0.561000; val_acc: 0.531000
(Iteration 4001 / 7350) loss: 1.508882
(Iteration 4101 / 7350) loss: 1.270624
(Iteration 4201 / 7350) loss: 1.370773
(Iteration 4301 / 7350) loss: 1.510028
(Iteration 4401 / 7350) loss: 1.357772
(Epoch 9 / 15) train acc: 0.551000; val_acc: 0.539000
(Iteration 4501 / 7350) loss: 1.429315
(Iteration 4601 / 7350) loss: 1.394038
(Iteration 4701 / 7350) loss: 1.301880
(Iteration 4801 / 7350) loss: 1.331816
(Epoch 10 / 15) train acc: 0.550000; val_acc: 0.540000
(Iteration 4901 / 7350) loss: 1.370802
(Iteration 5001 / 7350) loss: 1.266035
(Iteration 5101 / 7350) loss: 1.311282
(Iteration 5201 / 7350) loss: 1.295542
(Iteration 5301 / 7350) loss: 1.256411
(Epoch 11 / 15) train acc: 0.578000; val_acc: 0.544000
(Iteration 5401 / 7350) loss: 1.276002
(Iteration 5501 / 7350) loss: 1.348316
(Iteration 5601 / 7350) loss: 1.471356
(Iteration 5701 / 7350) loss: 1.317409
(Iteration 5801 / 7350) loss: 1.269281
(Epoch 12 / 15) train acc: 0.585000; val_acc: 0.548000
(Iteration 5901 / 7350) loss: 1.342864
(Iteration 6001 / 7350) loss: 1.248907
(Iteration 6101 / 7350) loss: 1.246018
```

```
(Iteration 6201 / 7350) loss: 1.347088
(Iteration 6301 / 7350) loss: 1.498109
(Epoch 13 / 15) train acc: 0.557000; val_acc: 0.542000
(Iteration 6401 / 7350) loss: 1.207681
(Iteration 6501 / 7350) loss: 1.136151
(Iteration 6601 / 7350) loss: 1.196962
(Iteration 6701 / 7350) loss: 1.339279
(Iteration 6801 / 7350) loss: 1.293082
(Epoch 14 / 15) train acc: 0.547000; val_acc: 0.558000
(Iteration 6901 / 7350) loss: 1.345312
(Iteration 7001 / 7350) loss: 1.261970
(Iteration 7101 / 7350) loss: 1.395386
(Iteration 7201 / 7350) loss: 1.155865
(Iteration 7301 / 7350) loss: 1.264296
(Epoch 15 / 15) train acc: 0.575000; val_acc: 0.553000
Best model not updated! Current best_val_acc = 0.606
Start computing trial 17/30...
reg: 0.0008273143279278817, lr: 0.025706714536697227, lr_decay:
0.9998464480802304
(Iteration 1 / 490) loss: 2.302639
(Epoch 0 / 1) train acc: 0.109000; val_acc: 0.093000
(Iteration 101 / 490) loss: 2.302289
(Iteration 201 / 490) loss: 2.297665
(Iteration 301 / 490) loss: 2.289747
(Iteration 401 / 490) loss: 2.230761
(Epoch 1 / 1) train acc: 0.239000; val_acc: 0.266000
Start computing trial 17/30...
reg: 0.0008273143279278817, lr: 0.025706714536697227, lr_decay:
0.9998464480802304
(Iteration 1 / 7350) loss: 2.302582
(Epoch 0 / 15) train acc: 0.155000; val_acc: 0.154000
(Iteration 101 / 7350) loss: 2.302675
(Iteration 201 / 7350) loss: 2.299029
(Iteration 301 / 7350) loss: 2.289490
(Iteration 401 / 7350) loss: 2.241221
(Epoch 1 / 15) train acc: 0.263000; val_acc: 0.281000
(Iteration 501 / 7350) loss: 2.107126
(Iteration 601 / 7350) loss: 2.039866
(Iteration 701 / 7350) loss: 1.845609
(Iteration 801 / 7350) loss: 1.820270
(Iteration 901 / 7350) loss: 1.761991
(Epoch 2 / 15) train acc: 0.436000; val_acc: 0.399000
(Iteration 1001 / 7350) loss: 1.675454
(Iteration 1101 / 7350) loss: 1.600946
(Iteration 1201 / 7350) loss: 1.740215
(Iteration 1301 / 7350) loss: 1.569418
(Iteration 1401 / 7350) loss: 1.489486
(Epoch 3 / 15) train acc: 0.468000; val_acc: 0.471000
```

```
(Iteration 1501 / 7350) loss: 1.395571
(Iteration 1601 / 7350) loss: 1.435214
(Iteration 1701 / 7350) loss: 1.565793
(Iteration 1801 / 7350) loss: 1.413125
(Iteration 1901 / 7350) loss: 1.360714
(Epoch 4 / 15) train acc: 0.507000; val_acc: 0.487000
(Iteration 2001 / 7350) loss: 1.349578
(Iteration 2101 / 7350) loss: 1.331515
(Iteration 2201 / 7350) loss: 1.341190
(Iteration 2301 / 7350) loss: 1.288158
(Iteration 2401 / 7350) loss: 1.308515
(Epoch 5 / 15) train acc: 0.520000; val_acc: 0.507000
(Iteration 2501 / 7350) loss: 1.442580
(Iteration 2601 / 7350) loss: 1.284772
(Iteration 2701 / 7350) loss: 1.287602
(Iteration 2801 / 7350) loss: 1.256855
(Iteration 2901 / 7350) loss: 1.473351
(Epoch 6 / 15) train acc: 0.525000; val_acc: 0.519000
(Iteration 3001 / 7350) loss: 1.483455
(Iteration 3101 / 7350) loss: 1.437506
(Iteration 3201 / 7350) loss: 1.540897
(Iteration 3301 / 7350) loss: 1.210768
(Iteration 3401 / 7350) loss: 1.204725
(Epoch 7 / 15) train acc: 0.515000; val_acc: 0.517000
(Iteration 3501 / 7350) loss: 1.261301
(Iteration 3601 / 7350) loss: 1.410047
(Iteration 3701 / 7350) loss: 1.229462
(Iteration 3801 / 7350) loss: 1.457770
(Iteration 3901 / 7350) loss: 1.374013
(Epoch 8 / 15) train acc: 0.555000; val_acc: 0.527000
(Iteration 4001 / 7350) loss: 1.563175
(Iteration 4101 / 7350) loss: 1.358884
(Iteration 4201 / 7350) loss: 1.246032
(Iteration 4301 / 7350) loss: 1.352098
(Iteration 4401 / 7350) loss: 1.334180
(Epoch 9 / 15) train acc: 0.530000; val_acc: 0.527000
(Iteration 4501 / 7350) loss: 1.250120
(Iteration 4601 / 7350) loss: 1.214465
(Iteration 4701 / 7350) loss: 1.302844
(Iteration 4801 / 7350) loss: 1.203747
(Epoch 10 / 15) train acc: 0.535000; val_acc: 0.528000
(Iteration 4901 / 7350) loss: 1.215417
(Iteration 5001 / 7350) loss: 1.347226
(Iteration 5101 / 7350) loss: 1.316687
(Iteration 5201 / 7350) loss: 1.304803
(Iteration 5301 / 7350) loss: 1.168649
(Epoch 11 / 15) train acc: 0.618000; val_acc: 0.542000
(Iteration 5401 / 7350) loss: 1.099226
```

```
(Iteration 5501 / 7350) loss: 1.114591
(Iteration 5601 / 7350) loss: 1.268792
(Iteration 5701 / 7350) loss: 1.385919
(Iteration 5801 / 7350) loss: 1.197396
(Epoch 12 / 15) train acc: 0.561000; val_acc: 0.541000
(Iteration 5901 / 7350) loss: 1.256998
(Iteration 6001 / 7350) loss: 1.211884
(Iteration 6101 / 7350) loss: 1.044499
(Iteration 6201 / 7350) loss: 1.262839
(Iteration 6301 / 7350) loss: 1.184739
(Epoch 13 / 15) train acc: 0.566000; val_acc: 0.543000
(Iteration 6401 / 7350) loss: 1.057850
(Iteration 6501 / 7350) loss: 1.186987
(Iteration 6601 / 7350) loss: 1.220603
(Iteration 6701 / 7350) loss: 0.929882
(Iteration 6801 / 7350) loss: 1.266703
(Epoch 14 / 15) train acc: 0.578000; val_acc: 0.557000
(Iteration 6901 / 7350) loss: 1.132388
(Iteration 7001 / 7350) loss: 1.198359
(Iteration 7101 / 7350) loss: 1.290870
(Iteration 7201 / 7350) loss: 1.212289
(Iteration 7301 / 7350) loss: 1.264021
(Epoch 15 / 15) train acc: 0.595000; val_acc: 0.556000
Best model not updated! Current best_val_acc = 0.606
Start computing trial 18/30...
reg: 0.005876499778752434, lr: 0.022799579993276718, lr_decay: 0.975570316469612
(Iteration 1 / 490) loss: 2.302821
(Epoch 0 / 1) train acc: 0.107000; val_acc: 0.100000
(Iteration 101 / 490) loss: 2.302281
(Iteration 201 / 490) loss: 2.299313
(Iteration 301 / 490) loss: 2.291372
(Iteration 401 / 490) loss: 2.268913
(Epoch 1 / 1) train acc: 0.275000; val_acc: 0.231000
Start computing trial 18/30...
reg: 0.005876499778752434, lr: 0.022799579993276718, lr_decay: 0.975570316469612
(Iteration 1 / 7350) loss: 2.302833
(Epoch 0 / 15) train acc: 0.114000; val_acc: 0.080000
(Iteration 101 / 7350) loss: 2.302916
(Iteration 201 / 7350) loss: 2.300163
(Iteration 301 / 7350) loss: 2.294630
(Iteration 401 / 7350) loss: 2.255056
(Epoch 1 / 15) train acc: 0.267000; val_acc: 0.258000
(Iteration 501 / 7350) loss: 2.203785
(Iteration 601 / 7350) loss: 2.042739
(Iteration 701 / 7350) loss: 2.053609
(Iteration 801 / 7350) loss: 1.897848
(Iteration 901 / 7350) loss: 1.849136
(Epoch 2 / 15) train acc: 0.386000; val_acc: 0.381000
```

```
(Iteration 1001 / 7350) loss: 1.693733
(Iteration 1101 / 7350) loss: 1.668105
(Iteration 1201 / 7350) loss: 1.585621
(Iteration 1301 / 7350) loss: 1.769371
(Iteration 1401 / 7350) loss: 1.626612
(Epoch 3 / 15) train acc: 0.459000; val_acc: 0.436000
(Iteration 1501 / 7350) loss: 1.574835
(Iteration 1601 / 7350) loss: 1.484802
(Iteration 1701 / 7350) loss: 1.448605
(Iteration 1801 / 7350) loss: 1.533029
(Iteration 1901 / 7350) loss: 1.557238
(Epoch 4 / 15) train acc: 0.480000; val_acc: 0.474000
(Iteration 2001 / 7350) loss: 1.449480
(Iteration 2101 / 7350) loss: 1.597194
(Iteration 2201 / 7350) loss: 1.401120
(Iteration 2301 / 7350) loss: 1.489349
(Iteration 2401 / 7350) loss: 1.426488
(Epoch 5 / 15) train acc: 0.478000; val_acc: 0.498000
(Iteration 2501 / 7350) loss: 1.286558
(Iteration 2601 / 7350) loss: 1.417731
(Iteration 2701 / 7350) loss: 1.377300
(Iteration 2801 / 7350) loss: 1.564587
(Iteration 2901 / 7350) loss: 1.592450
(Epoch 6 / 15) train acc: 0.531000; val_acc: 0.512000
(Iteration 3001 / 7350) loss: 1.434633
(Iteration 3101 / 7350) loss: 1.400064
(Iteration 3201 / 7350) loss: 1.555546
(Iteration 3301 / 7350) loss: 1.335312
(Iteration 3401 / 7350) loss: 1.302241
(Epoch 7 / 15) train acc: 0.491000; val_acc: 0.510000
(Iteration 3501 / 7350) loss: 1.389956
(Iteration 3601 / 7350) loss: 1.444174
(Iteration 3701 / 7350) loss: 1.360407
(Iteration 3801 / 7350) loss: 1.438260
(Iteration 3901 / 7350) loss: 1.575516
(Epoch 8 / 15) train acc: 0.527000; val_acc: 0.522000
(Iteration 4001 / 7350) loss: 1.414386
(Iteration 4101 / 7350) loss: 1.293477
(Iteration 4201 / 7350) loss: 1.353678
(Iteration 4301 / 7350) loss: 1.280111
(Iteration 4401 / 7350) loss: 1.484195
(Epoch 9 / 15) train acc: 0.496000; val_acc: 0.515000
(Iteration 4501 / 7350) loss: 1.301626
(Iteration 4601 / 7350) loss: 1.511728
(Iteration 4701 / 7350) loss: 1.391024
(Iteration 4801 / 7350) loss: 1.480851
(Epoch 10 / 15) train acc: 0.535000; val_acc: 0.520000
(Iteration 4901 / 7350) loss: 1.415106
```

```
(Iteration 5001 / 7350) loss: 1.232667
(Iteration 5101 / 7350) loss: 1.337729
(Iteration 5201 / 7350) loss: 1.494779
(Iteration 5301 / 7350) loss: 1.441620
(Epoch 11 / 15) train acc: 0.516000; val_acc: 0.524000
(Iteration 5401 / 7350) loss: 1.525787
(Iteration 5501 / 7350) loss: 1.475582
(Iteration 5601 / 7350) loss: 1.447619
(Iteration 5701 / 7350) loss: 1.326039
(Iteration 5801 / 7350) loss: 1.422589
(Epoch 12 / 15) train acc: 0.524000; val_acc: 0.521000
(Iteration 5901 / 7350) loss: 1.434120
(Iteration 6001 / 7350) loss: 1.531295
(Iteration 6101 / 7350) loss: 1.474982
(Iteration 6201 / 7350) loss: 1.400348
(Iteration 6301 / 7350) loss: 1.384640
(Epoch 13 / 15) train acc: 0.553000; val_acc: 0.527000
(Iteration 6401 / 7350) loss: 1.504510
(Iteration 6501 / 7350) loss: 1.333237
(Iteration 6601 / 7350) loss: 1.326199
(Iteration 6701 / 7350) loss: 1.469660
(Iteration 6801 / 7350) loss: 1.351116
(Epoch 14 / 15) train acc: 0.544000; val_acc: 0.525000
(Iteration 6901 / 7350) loss: 1.471254
(Iteration 7001 / 7350) loss: 1.308846
(Iteration 7101 / 7350) loss: 1.202759
(Iteration 7201 / 7350) loss: 1.307930
(Iteration 7301 / 7350) loss: 1.382743
(Epoch 15 / 15) train acc: 0.546000; val_acc: 0.532000
Best model not updated! Current best_val_acc = 0.606
Start computing trial 19/30...
reg: 0.004763550943491087, lr: 0.05784096999687347, lr_decay: 0.9812009131130642
(Iteration 1 / 490) loss: 2.302727
(Epoch 0 / 1) train acc: 0.096000; val_acc: 0.113000
(Iteration 101 / 490) loss: 2.298457
(Iteration 201 / 490) loss: 2.192091
(Iteration 301 / 490) loss: 1.848255
(Iteration 401 / 490) loss: 1.649141
(Epoch 1 / 1) train acc: 0.454000; val_acc: 0.403000
Start computing trial 19/30...
reg: 0.004763550943491087, lr: 0.05784096999687347, lr_decay: 0.9812009131130642
(Iteration 1 / 7350) loss: 2.302794
(Epoch 0 / 15) train acc: 0.106000; val_acc: 0.086000
(Iteration 101 / 7350) loss: 2.295113
(Iteration 201 / 7350) loss: 2.165571
(Iteration 301 / 7350) loss: 1.921262
(Iteration 401 / 7350) loss: 1.722319
(Epoch 1 / 15) train acc: 0.441000; val_acc: 0.420000
```

```
(Iteration 501 / 7350) loss: 1.556199
(Iteration 601 / 7350) loss: 1.517821
(Iteration 701 / 7350) loss: 1.465831
(Iteration 801 / 7350) loss: 1.443794
(Iteration 901 / 7350) loss: 1.473402
(Epoch 2 / 15) train acc: 0.513000; val_acc: 0.506000
(Iteration 1001 / 7350) loss: 1.449555
(Iteration 1101 / 7350) loss: 1.597738
(Iteration 1201 / 7350) loss: 1.465705
(Iteration 1301 / 7350) loss: 1.477274
(Iteration 1401 / 7350) loss: 1.315789
(Epoch 3 / 15) train acc: 0.485000; val_acc: 0.504000
(Iteration 1501 / 7350) loss: 1.429610
(Iteration 1601 / 7350) loss: 1.523045
(Iteration 1701 / 7350) loss: 1.176472
(Iteration 1801 / 7350) loss: 1.273600
(Iteration 1901 / 7350) loss: 1.534589
(Epoch 4 / 15) train acc: 0.538000; val_acc: 0.516000
(Iteration 2001 / 7350) loss: 1.367327
(Iteration 2101 / 7350) loss: 1.279144
(Iteration 2201 / 7350) loss: 1.369810
(Iteration 2301 / 7350) loss: 1.493947
(Iteration 2401 / 7350) loss: 1.419250
(Epoch 5 / 15) train acc: 0.545000; val_acc: 0.525000
(Iteration 2501 / 7350) loss: 1.504862
(Iteration 2601 / 7350) loss: 1.298230
(Iteration 2701 / 7350) loss: 1.435953
(Iteration 2801 / 7350) loss: 1.268594
(Iteration 2901 / 7350) loss: 1.297818
(Epoch 6 / 15) train acc: 0.567000; val_acc: 0.527000
(Iteration 3001 / 7350) loss: 1.599975
(Iteration 3101 / 7350) loss: 1.362014
(Iteration 3201 / 7350) loss: 1.259120
(Iteration 3301 / 7350) loss: 1.424370
(Iteration 3401 / 7350) loss: 1.515641
(Epoch 7 / 15) train acc: 0.557000; val_acc: 0.538000
(Iteration 3501 / 7350) loss: 1.242094
(Iteration 3601 / 7350) loss: 1.371648
(Iteration 3701 / 7350) loss: 1.366435
(Iteration 3801 / 7350) loss: 1.288317
(Iteration 3901 / 7350) loss: 1.222672
(Epoch 8 / 15) train acc: 0.563000; val_acc: 0.543000
(Iteration 4001 / 7350) loss: 1.298694
(Iteration 4101 / 7350) loss: 1.302215
(Iteration 4201 / 7350) loss: 1.154737
(Iteration 4301 / 7350) loss: 1.193021
(Iteration 4401 / 7350) loss: 1.335431
(Epoch 9 / 15) train acc: 0.573000; val_acc: 0.539000
```

```
(Iteration 4501 / 7350) loss: 1.425374
(Iteration 4601 / 7350) loss: 1.286462
(Iteration 4701 / 7350) loss: 1.247063
(Iteration 4801 / 7350) loss: 1.219845
(Epoch 10 / 15) train acc: 0.572000; val_acc: 0.549000
(Iteration 4901 / 7350) loss: 1.082582
(Iteration 5001 / 7350) loss: 1.442784
(Iteration 5101 / 7350) loss: 1.149817
(Iteration 5201 / 7350) loss: 1.284770
(Iteration 5301 / 7350) loss: 1.183251
(Epoch 11 / 15) train acc: 0.561000; val_acc: 0.560000
(Iteration 5401 / 7350) loss: 1.270385
(Iteration 5501 / 7350) loss: 1.346828
(Iteration 5601 / 7350) loss: 1.371133
(Iteration 5701 / 7350) loss: 1.210307
(Iteration 5801 / 7350) loss: 1.350270
(Epoch 12 / 15) train acc: 0.602000; val_acc: 0.565000
(Iteration 5901 / 7350) loss: 1.222880
(Iteration 6001 / 7350) loss: 1.265247
(Iteration 6101 / 7350) loss: 1.473322
(Iteration 6201 / 7350) loss: 1.146230
(Iteration 6301 / 7350) loss: 1.481925
(Epoch 13 / 15) train acc: 0.600000; val_acc: 0.568000
(Iteration 6401 / 7350) loss: 1.348586
(Iteration 6501 / 7350) loss: 1.291138
(Iteration 6601 / 7350) loss: 1.096483
(Iteration 6701 / 7350) loss: 1.136669
(Iteration 6801 / 7350) loss: 1.296387
(Epoch 14 / 15) train acc: 0.594000; val_acc: 0.576000
(Iteration 6901 / 7350) loss: 1.195788
(Iteration 7001 / 7350) loss: 1.075392
(Iteration 7101 / 7350) loss: 1.285958
(Iteration 7201 / 7350) loss: 1.438361
(Iteration 7301 / 7350) loss: 1.208510
(Epoch 15 / 15) train acc: 0.623000; val_acc: 0.564000
Best model not updated! Current best_val_acc = 0.606
Start computing trial 20/30...
reg: 0.003269320070203724, lr: 0.013051583120246308, lr_decay:
0.9857915819229627
(Iteration 1 / 490) loss: 2.302730
(Epoch 0 / 1) train acc: 0.113000; val_acc: 0.120000
(Iteration 101 / 490) loss: 2.302772
(Iteration 201 / 490) loss: 2.301367
(Iteration 301 / 490) loss: 2.301116
(Iteration 401 / 490) loss: 2.300875
(Epoch 1 / 1) train acc: 0.225000; val_acc: 0.212000
Start computing trial 20/30...
reg: 0.003269320070203724, lr: 0.013051583120246308, lr_decay:
```

```
0.9857915819229627
(Iteration 1 / 7350) loss: 2.302690
(Epoch 0 / 15) train acc: 0.113000; val_acc: 0.081000
(Iteration 101 / 7350) loss: 2.303319
(Iteration 201 / 7350) loss: 2.301937
(Iteration 301 / 7350) loss: 2.301967
(Iteration 401 / 7350) loss: 2.300090
(Epoch 1 / 15) train acc: 0.292000; val_acc: 0.310000
(Iteration 501 / 7350) loss: 2.298085
(Iteration 601 / 7350) loss: 2.288744
(Iteration 701 / 7350) loss: 2.271908
(Iteration 801 / 7350) loss: 2.236987
(Iteration 901 / 7350) loss: 2.172227
(Epoch 2 / 15) train acc: 0.242000; val_acc: 0.262000
(Iteration 1001 / 7350) loss: 2.124515
(Iteration 1101 / 7350) loss: 2.029611
(Iteration 1201 / 7350) loss: 1.990271
(Iteration 1301 / 7350) loss: 1.830778
(Iteration 1401 / 7350) loss: 1.817221
(Epoch 3 / 15) train acc: 0.327000; val_acc: 0.343000
(Iteration 1501 / 7350) loss: 1.831182
(Iteration 1601 / 7350) loss: 1.720367
(Iteration 1701 / 7350) loss: 1.736010
(Iteration 1801 / 7350) loss: 1.631942
(Iteration 1901 / 7350) loss: 1.692082
(Epoch 4 / 15) train acc: 0.419000; val_acc: 0.406000
(Iteration 2001 / 7350) loss: 1.734235
(Iteration 2101 / 7350) loss: 1.682038
(Iteration 2201 / 7350) loss: 1.551316
(Iteration 2301 / 7350) loss: 1.540655
(Iteration 2401 / 7350) loss: 1.539799
(Epoch 5 / 15) train acc: 0.465000; val_acc: 0.433000
(Iteration 2501 / 7350) loss: 1.624251
(Iteration 2601 / 7350) loss: 1.530067
(Iteration 2701 / 7350) loss: 1.592339
(Iteration 2801 / 7350) loss: 1.366506
(Iteration 2901 / 7350) loss: 1.368146
(Epoch 6 / 15) train acc: 0.473000; val_acc: 0.466000
(Iteration 3001 / 7350) loss: 1.434814
(Iteration 3101 / 7350) loss: 1.456718
(Iteration 3201 / 7350) loss: 1.471096
(Iteration 3301 / 7350) loss: 1.480093
(Iteration 3401 / 7350) loss: 1.408327
(Epoch 7 / 15) train acc: 0.488000; val_acc: 0.483000
(Iteration 3501 / 7350) loss: 1.430159
(Iteration 3601 / 7350) loss: 1.435919
(Iteration 3701 / 7350) loss: 1.401689
(Iteration 3801 / 7350) loss: 1.325462
```

```
(Iteration 3901 / 7350) loss: 1.586574
(Epoch 8 / 15) train acc: 0.479000; val_acc: 0.496000
(Iteration 4001 / 7350) loss: 1.592110
(Iteration 4101 / 7350) loss: 1.304957
(Iteration 4201 / 7350) loss: 1.491752
(Iteration 4301 / 7350) loss: 1.426237
(Iteration 4401 / 7350) loss: 1.477906
(Epoch 9 / 15) train acc: 0.486000; val_acc: 0.511000
(Iteration 4501 / 7350) loss: 1.468006
(Iteration 4601 / 7350) loss: 1.475149
(Iteration 4701 / 7350) loss: 1.527549
(Iteration 4801 / 7350) loss: 1.469634
(Epoch 10 / 15) train acc: 0.506000; val_acc: 0.515000
(Iteration 4901 / 7350) loss: 1.322023
(Iteration 5001 / 7350) loss: 1.384389
(Iteration 5101 / 7350) loss: 1.260320
(Iteration 5201 / 7350) loss: 1.723999
(Iteration 5301 / 7350) loss: 1.389171
(Epoch 11 / 15) train acc: 0.538000; val_acc: 0.515000
(Iteration 5401 / 7350) loss: 1.685193
(Iteration 5501 / 7350) loss: 1.531281
(Iteration 5601 / 7350) loss: 1.328245
(Iteration 5701 / 7350) loss: 1.452083
(Iteration 5801 / 7350) loss: 1.413096
(Epoch 12 / 15) train acc: 0.541000; val_acc: 0.519000
(Iteration 5901 / 7350) loss: 1.626864
(Iteration 6001 / 7350) loss: 1.412262
(Iteration 6101 / 7350) loss: 1.285704
(Iteration 6201 / 7350) loss: 1.381821
(Iteration 6301 / 7350) loss: 1.443433
(Epoch 13 / 15) train acc: 0.528000; val_acc: 0.519000
(Iteration 6401 / 7350) loss: 1.244434
(Iteration 6501 / 7350) loss: 1.445063
(Iteration 6601 / 7350) loss: 1.439607
(Iteration 6701 / 7350) loss: 1.395619
(Iteration 6801 / 7350) loss: 1.304296
(Epoch 14 / 15) train acc: 0.553000; val_acc: 0.519000
(Iteration 6901 / 7350) loss: 1.288254
(Iteration 7001 / 7350) loss: 1.243807
(Iteration 7101 / 7350) loss: 1.472015
(Iteration 7201 / 7350) loss: 1.392607
(Iteration 7301 / 7350) loss: 1.244126
(Epoch 15 / 15) train acc: 0.509000; val_acc: 0.520000
Best model not updated! Current best_val_acc = 0.606
Start computing trial 21/30...
reg: 0.006930120725478431, lr: 0.05431479850855715, lr_decay: 0.9877637856775213
(Iteration 1 / 490) loss: 2.302863
(Epoch 0 / 1) train acc: 0.092000; val_acc: 0.086000
```

```
(Iteration 101 / 490) loss: 2.297551
(Iteration 201 / 490) loss: 2.210269
(Iteration 301 / 490) loss: 2.015909
(Iteration 401 / 490) loss: 1.761503
(Epoch 1 / 1) train acc: 0.405000; val_acc: 0.404000
Start computing trial 21/30...
reg: 0.006930120725478431, lr: 0.05431479850855715, lr_decay: 0.9877637856775213
(Iteration 1 / 7350) loss: 2.302892
(Epoch 0 / 15) train acc: 0.104000; val_acc: 0.087000
(Iteration 101 / 7350) loss: 2.297017
(Iteration 201 / 7350) loss: 2.229436
(Iteration 301 / 7350) loss: 2.065486
(Iteration 401 / 7350) loss: 1.773934
(Epoch 1 / 15) train acc: 0.404000; val_acc: 0.399000
(Iteration 501 / 7350) loss: 1.702012
(Iteration 601 / 7350) loss: 1.557179
(Iteration 701 / 7350) loss: 1.487256
(Iteration 801 / 7350) loss: 1.526954
(Iteration 901 / 7350) loss: 1.480050
(Epoch 2 / 15) train acc: 0.506000; val_acc: 0.495000
(Iteration 1001 / 7350) loss: 1.583557
(Iteration 1101 / 7350) loss: 1.423053
(Iteration 1201 / 7350) loss: 1.347041
(Iteration 1301 / 7350) loss: 1.320561
(Iteration 1401 / 7350) loss: 1.513052
(Epoch 3 / 15) train acc: 0.521000; val_acc: 0.509000
(Iteration 1501 / 7350) loss: 1.582526
(Iteration 1601 / 7350) loss: 1.619622
(Iteration 1701 / 7350) loss: 1.368453
(Iteration 1801 / 7350) loss: 1.430718
(Iteration 1901 / 7350) loss: 1.518324
(Epoch 4 / 15) train acc: 0.522000; val_acc: 0.515000
(Iteration 2001 / 7350) loss: 1.355684
(Iteration 2101 / 7350) loss: 1.427485
(Iteration 2201 / 7350) loss: 1.327967
(Iteration 2301 / 7350) loss: 1.532255
(Iteration 2401 / 7350) loss: 1.438058
(Epoch 5 / 15) train acc: 0.547000; val_acc: 0.527000
(Iteration 2501 / 7350) loss: 1.536937
(Iteration 2601 / 7350) loss: 1.332257
(Iteration 2701 / 7350) loss: 1.472883
(Iteration 2801 / 7350) loss: 1.427547
(Iteration 2901 / 7350) loss: 1.328663
(Epoch 6 / 15) train acc: 0.570000; val_acc: 0.539000
(Iteration 3001 / 7350) loss: 1.412220
(Iteration 3101 / 7350) loss: 1.406814
(Iteration 3201 / 7350) loss: 1.343297
(Iteration 3301 / 7350) loss: 1.188914
```

```
(Iteration 3401 / 7350) loss: 1.516260
(Epoch 7 / 15) train acc: 0.536000; val_acc: 0.531000
(Iteration 3501 / 7350) loss: 1.379276
(Iteration 3601 / 7350) loss: 1.525889
(Iteration 3701 / 7350) loss: 1.474044
(Iteration 3801 / 7350) loss: 1.488340
(Iteration 3901 / 7350) loss: 1.444875
(Epoch 8 / 15) train acc: 0.558000; val_acc: 0.531000
(Iteration 4001 / 7350) loss: 1.336383
(Iteration 4101 / 7350) loss: 1.316682
(Iteration 4201 / 7350) loss: 1.404029
(Iteration 4301 / 7350) loss: 1.562566
(Iteration 4401 / 7350) loss: 1.355175
(Epoch 9 / 15) train acc: 0.578000; val_acc: 0.546000
(Iteration 4501 / 7350) loss: 1.363444
(Iteration 4601 / 7350) loss: 1.364604
(Iteration 4701 / 7350) loss: 1.314517
(Iteration 4801 / 7350) loss: 1.323847
(Epoch 10 / 15) train acc: 0.574000; val_acc: 0.559000
(Iteration 4901 / 7350) loss: 1.399571
(Iteration 5001 / 7350) loss: 1.369668
(Iteration 5101 / 7350) loss: 1.298282
(Iteration 5201 / 7350) loss: 1.308485
(Iteration 5301 / 7350) loss: 1.381886
(Epoch 11 / 15) train acc: 0.561000; val_acc: 0.553000
(Iteration 5401 / 7350) loss: 1.497874
(Iteration 5501 / 7350) loss: 1.370911
(Iteration 5601 / 7350) loss: 1.391679
(Iteration 5701 / 7350) loss: 1.355868
(Iteration 5801 / 7350) loss: 1.408793
(Epoch 12 / 15) train acc: 0.606000; val_acc: 0.547000
(Iteration 5901 / 7350) loss: 1.162951
(Iteration 6001 / 7350) loss: 1.290411
(Iteration 6101 / 7350) loss: 1.493157
(Iteration 6201 / 7350) loss: 1.350133
(Iteration 6301 / 7350) loss: 1.320223
(Epoch 13 / 15) train acc: 0.587000; val_acc: 0.554000
(Iteration 6401 / 7350) loss: 1.272170
(Iteration 6501 / 7350) loss: 1.271848
(Iteration 6601 / 7350) loss: 1.257740
(Iteration 6701 / 7350) loss: 1.340035
(Iteration 6801 / 7350) loss: 1.471048
(Epoch 14 / 15) train acc: 0.593000; val_acc: 0.557000
(Iteration 6901 / 7350) loss: 1.314154
(Iteration 7001 / 7350) loss: 1.331156
(Iteration 7101 / 7350) loss: 1.362000
(Iteration 7201 / 7350) loss: 1.269351
(Iteration 7301 / 7350) loss: 1.629715
```

```
(Epoch 15 / 15) train acc: 0.612000; val_acc: 0.565000
Best model not updated! Current best_val_acc = 0.606
Start computing trial 22/30...
reg: 0.006634483473628818, lr: 0.010126900733574145, lr_decay:
0.9951000767668773
(Iteration 1 / 490) loss: 2.302877
(Epoch 0 / 1) train acc: 0.079000; val_acc: 0.118000
(Iteration 101 / 490) loss: 2.302593
(Iteration 201 / 490) loss: 2.302773
(Iteration 301 / 490) loss: 2.302135
(Iteration 401 / 490) loss: 2.300937
(Epoch 1 / 1) train acc: 0.121000; val_acc: 0.114000
Start computing trial 23/30...
reg: 0.004981125924755854, lr: 0.017762962480430324, lr_decay:
0.9941698217349275
(Iteration 1 / 490) loss: 2.302800
(Epoch 0 / 1) train acc: 0.114000; val_acc: 0.084000
(Iteration 101 / 490) loss: 2.302057
(Iteration 201 / 490) loss: 2.300321
(Iteration 301 / 490) loss: 2.299156
(Iteration 401 / 490) loss: 2.291951
(Epoch 1 / 1) train acc: 0.262000; val_acc: 0.276000
Start computing trial 23/30...
reg: 0.004981125924755854, lr: 0.017762962480430324, lr_decay:
0.9941698217349275
(Iteration 1 / 7350) loss: 2.302822
(Epoch 0 / 15) train acc: 0.117000; val_acc: 0.137000
(Iteration 101 / 7350) loss: 2.302347
(Iteration 201 / 7350) loss: 2.302253
(Iteration 301 / 7350) loss: 2.299246
(Iteration 401 / 7350) loss: 2.292530
(Epoch 1 / 15) train acc: 0.305000; val_acc: 0.302000
(Iteration 501 / 7350) loss: 2.274744
(Iteration 601 / 7350) loss: 2.227896
(Iteration 701 / 7350) loss: 2.155494
(Iteration 801 / 7350) loss: 1.986912
(Iteration 901 / 7350) loss: 1.921485
(Epoch 2 / 15) train acc: 0.302000; val_acc: 0.320000
(Iteration 1001 / 7350) loss: 1.917316
(Iteration 1101 / 7350) loss: 1.868174
(Iteration 1201 / 7350) loss: 1.740895
(Iteration 1301 / 7350) loss: 1.667587
(Iteration 1401 / 7350) loss: 1.541799
(Epoch 3 / 15) train acc: 0.427000; val_acc: 0.407000
(Iteration 1501 / 7350) loss: 1.667201
(Iteration 1601 / 7350) loss: 1.607192
(Iteration 1701 / 7350) loss: 1.510369
(Iteration 1801 / 7350) loss: 1.622043
```

```
(Iteration 1901 / 7350) loss: 1.586186
(Epoch 4 / 15) train acc: 0.468000; val_acc: 0.441000
(Iteration 2001 / 7350) loss: 1.520160
(Iteration 2101 / 7350) loss: 1.638200
(Iteration 2201 / 7350) loss: 1.606090
(Iteration 2301 / 7350) loss: 1.554421
(Iteration 2401 / 7350) loss: 1.379832
(Epoch 5 / 15) train acc: 0.487000; val_acc: 0.486000
(Iteration 2501 / 7350) loss: 1.380768
(Iteration 2601 / 7350) loss: 1.386027
(Iteration 2701 / 7350) loss: 1.562299
(Iteration 2801 / 7350) loss: 1.324330
(Iteration 2901 / 7350) loss: 1.504026
(Epoch 6 / 15) train acc: 0.506000; val_acc: 0.501000
(Iteration 3001 / 7350) loss: 1.473270
(Iteration 3101 / 7350) loss: 1.440486
(Iteration 3201 / 7350) loss: 1.523786
(Iteration 3301 / 7350) loss: 1.469223
(Iteration 3401 / 7350) loss: 1.521170
(Epoch 7 / 15) train acc: 0.516000; val_acc: 0.509000
(Iteration 3501 / 7350) loss: 1.488958
(Iteration 3601 / 7350) loss: 1.408500
(Iteration 3701 / 7350) loss: 1.372890
(Iteration 3801 / 7350) loss: 1.668711
(Iteration 3901 / 7350) loss: 1.401357
(Epoch 8 / 15) train acc: 0.514000; val_acc: 0.512000
(Iteration 4001 / 7350) loss: 1.502138
(Iteration 4101 / 7350) loss: 1.330232
(Iteration 4201 / 7350) loss: 1.532846
(Iteration 4301 / 7350) loss: 1.324811
(Iteration 4401 / 7350) loss: 1.404985
(Epoch 9 / 15) train acc: 0.535000; val_acc: 0.510000
(Iteration 4501 / 7350) loss: 1.371950
(Iteration 4601 / 7350) loss: 1.373307
(Iteration 4701 / 7350) loss: 1.249431
(Iteration 4801 / 7350) loss: 1.481953
(Epoch 10 / 15) train acc: 0.539000; val_acc: 0.514000
(Iteration 4901 / 7350) loss: 1.408164
(Iteration 5001 / 7350) loss: 1.534074
(Iteration 5101 / 7350) loss: 1.441536
(Iteration 5201 / 7350) loss: 1.349051
(Iteration 5301 / 7350) loss: 1.499677
(Epoch 11 / 15) train acc: 0.554000; val_acc: 0.517000
(Iteration 5401 / 7350) loss: 1.336261
(Iteration 5501 / 7350) loss: 1.332041
(Iteration 5601 / 7350) loss: 1.238571
(Iteration 5701 / 7350) loss: 1.426965
(Iteration 5801 / 7350) loss: 1.433262
```

```
(Epoch 12 / 15) train acc: 0.547000; val_acc: 0.522000
(Iteration 5901 / 7350) loss: 1.410396
(Iteration 6001 / 7350) loss: 1.417682
(Iteration 6101 / 7350) loss: 1.298814
(Iteration 6201 / 7350) loss: 1.337685
(Iteration 6301 / 7350) loss: 1.376405
(Epoch 13 / 15) train acc: 0.558000; val_acc: 0.529000
(Iteration 6401 / 7350) loss: 1.304395
(Iteration 6501 / 7350) loss: 1.470374
(Iteration 6601 / 7350) loss: 1.059833
(Iteration 6701 / 7350) loss: 1.430615
(Iteration 6801 / 7350) loss: 1.390208
(Epoch 14 / 15) train acc: 0.534000; val_acc: 0.528000
(Iteration 6901 / 7350) loss: 1.390551
(Iteration 7001 / 7350) loss: 1.504583
(Iteration 7101 / 7350) loss: 1.469241
(Iteration 7201 / 7350) loss: 1.341673
(Iteration 7301 / 7350) loss: 1.268973
(Epoch 15 / 15) train acc: 0.555000; val_acc: 0.534000
Best model not updated! Current best_val_acc = 0.606
Start computing trial 24/30...
reg: 0.000685428407906481, lr: 0.015534540308882253, lr_decay:
0.9883669547990624
(Iteration 1 / 490) loss: 2.302597
(Epoch 0 / 1) train acc: 0.108000; val_acc: 0.113000
(Iteration 101 / 490) loss: 2.302550
(Iteration 201 / 490) loss: 2.300853
(Iteration 301 / 490) loss: 2.300557
(Iteration 401 / 490) loss: 2.298179
(Epoch 1 / 1) train acc: 0.226000; val_acc: 0.263000
Start computing trial 24/30...
reg: 0.000685428407906481, lr: 0.015534540308882253, lr_decay:
0.9883669547990624
(Iteration 1 / 7350) loss: 2.302619
(Epoch 0 / 15) train acc: 0.092000; val_acc: 0.080000
(Iteration 101 / 7350) loss: 2.302256
(Iteration 201 / 7350) loss: 2.302533
(Iteration 301 / 7350) loss: 2.300264
(Iteration 401 / 7350) loss: 2.295817
(Epoch 1 / 15) train acc: 0.293000; val_acc: 0.309000
(Iteration 501 / 7350) loss: 2.287999
(Iteration 601 / 7350) loss: 2.246232
(Iteration 701 / 7350) loss: 2.219072
(Iteration 801 / 7350) loss: 2.081958
(Iteration 901 / 7350) loss: 2.033327
(Epoch 2 / 15) train acc: 0.281000; val_acc: 0.288000
(Iteration 1001 / 7350) loss: 1.971276
(Iteration 1101 / 7350) loss: 1.882611
```

```
(Iteration 1201 / 7350) loss: 1.886504
(Iteration 1301 / 7350) loss: 1.672114
(Iteration 1401 / 7350) loss: 1.724641
(Epoch 3 / 15) train acc: 0.379000; val_acc: 0.390000
(Iteration 1501 / 7350) loss: 1.730462
(Iteration 1601 / 7350) loss: 1.701255
(Iteration 1701 / 7350) loss: 1.580256
(Iteration 1801 / 7350) loss: 1.772890
(Iteration 1901 / 7350) loss: 1.690127
(Epoch 4 / 15) train acc: 0.456000; val_acc: 0.427000
(Iteration 2001 / 7350) loss: 1.477259
(Iteration 2101 / 7350) loss: 1.656610
(Iteration 2201 / 7350) loss: 1.498957
(Iteration 2301 / 7350) loss: 1.297992
(Iteration 2401 / 7350) loss: 1.438589
(Epoch 5 / 15) train acc: 0.473000; val_acc: 0.471000
(Iteration 2501 / 7350) loss: 1.494291
(Iteration 2601 / 7350) loss: 1.328100
(Iteration 2701 / 7350) loss: 1.456339
(Iteration 2801 / 7350) loss: 1.357865
(Iteration 2901 / 7350) loss: 1.428284
(Epoch 6 / 15) train acc: 0.506000; val_acc: 0.487000
(Iteration 3001 / 7350) loss: 1.450905
(Iteration 3101 / 7350) loss: 1.336976
(Iteration 3201 / 7350) loss: 1.307676
(Iteration 3301 / 7350) loss: 1.524625
(Iteration 3401 / 7350) loss: 1.457986
(Epoch 7 / 15) train acc: 0.501000; val_acc: 0.505000
(Iteration 3501 / 7350) loss: 1.338142
(Iteration 3601 / 7350) loss: 1.307013
(Iteration 3701 / 7350) loss: 1.298638
(Iteration 3801 / 7350) loss: 1.475963
(Iteration 3901 / 7350) loss: 1.235689
(Epoch 8 / 15) train acc: 0.554000; val_acc: 0.511000
(Iteration 4001 / 7350) loss: 1.402713
(Iteration 4101 / 7350) loss: 1.436401
(Iteration 4201 / 7350) loss: 1.456048
(Iteration 4301 / 7350) loss: 1.367504
(Iteration 4401 / 7350) loss: 1.339349
(Epoch 9 / 15) train acc: 0.514000; val_acc: 0.518000
(Iteration 4501 / 7350) loss: 1.470555
(Iteration 4601 / 7350) loss: 1.528753
(Iteration 4701 / 7350) loss: 1.275245
(Iteration 4801 / 7350) loss: 1.254064
(Epoch 10 / 15) train acc: 0.533000; val_acc: 0.512000
(Iteration 4901 / 7350) loss: 1.331632
(Iteration 5001 / 7350) loss: 1.279084
(Iteration 5101 / 7350) loss: 1.183182
```

```
(Iteration 5201 / 7350) loss: 1.332813
(Iteration 5301 / 7350) loss: 1.275549
(Epoch 11 / 15) train acc: 0.537000; val_acc: 0.520000
(Iteration 5401 / 7350) loss: 1.227100
(Iteration 5501 / 7350) loss: 1.495048
(Iteration 5601 / 7350) loss: 1.179823
(Iteration 5701 / 7350) loss: 1.528961
(Iteration 5801 / 7350) loss: 1.258593
(Epoch 12 / 15) train acc: 0.533000; val_acc: 0.520000
(Iteration 5901 / 7350) loss: 1.529311
(Iteration 6001 / 7350) loss: 1.328216
(Iteration 6101 / 7350) loss: 1.357030
(Iteration 6201 / 7350) loss: 1.465548
(Iteration 6301 / 7350) loss: 1.262206
(Epoch 13 / 15) train acc: 0.548000; val_acc: 0.524000
(Iteration 6401 / 7350) loss: 1.436618
(Iteration 6501 / 7350) loss: 1.155861
(Iteration 6601 / 7350) loss: 1.347875
(Iteration 6701 / 7350) loss: 1.247612
(Iteration 6801 / 7350) loss: 1.402726
(Epoch 14 / 15) train acc: 0.555000; val_acc: 0.522000
(Iteration 6901 / 7350) loss: 1.374069
(Iteration 7001 / 7350) loss: 1.368609
(Iteration 7101 / 7350) loss: 1.348862
(Iteration 7201 / 7350) loss: 1.359669
(Iteration 7301 / 7350) loss: 1.480006
(Epoch 15 / 15) train acc: 0.543000; val_acc: 0.534000
Best model not updated! Current best_val_acc = 0.606
Start computing trial 25/30...
reg: 0.008993601573174841, lr: 0.008669844752504366, lr_decay: 0.992052859299905
(Iteration 1 / 490) loss: 2.302930
(Epoch 0 / 1) train acc: 0.116000; val_acc: 0.136000
(Iteration 101 / 490) loss: 2.303436
(Iteration 201 / 490) loss: 2.302329
(Iteration 301 / 490) loss: 2.302070
(Iteration 401 / 490) loss: 2.302023
(Epoch 1 / 1) train acc: 0.189000; val_acc: 0.213000
Start computing trial 25/30...
reg: 0.008993601573174841, lr: 0.008669844752504366, lr_decay: 0.992052859299905
(Iteration 1 / 7350) loss: 2.302959
(Epoch 0 / 15) train acc: 0.080000; val_acc: 0.084000
(Iteration 101 / 7350) loss: 2.302739
(Iteration 201 / 7350) loss: 2.302613
(Iteration 301 / 7350) loss: 2.302232
(Iteration 401 / 7350) loss: 2.301836
(Epoch 1 / 15) train acc: 0.172000; val_acc: 0.154000
(Iteration 501 / 7350) loss: 2.301357
(Iteration 601 / 7350) loss: 2.299895
```

```
(Iteration 701 / 7350) loss: 2.298811
(Iteration 801 / 7350) loss: 2.295893
(Iteration 901 / 7350) loss: 2.285093
(Epoch 2 / 15) train acc: 0.214000; val_acc: 0.205000
(Iteration 1001 / 7350) loss: 2.281392
(Iteration 1101 / 7350) loss: 2.274484
(Iteration 1201 / 7350) loss: 2.241104
(Iteration 1301 / 7350) loss: 2.182567
(Iteration 1401 / 7350) loss: 2.163596
(Epoch 3 / 15) train acc: 0.252000; val_acc: 0.251000
(Iteration 1501 / 7350) loss: 2.149360
(Iteration 1601 / 7350) loss: 2.013045
(Iteration 1701 / 7350) loss: 1.997490
(Iteration 1801 / 7350) loss: 1.991824
(Iteration 1901 / 7350) loss: 1.939642
(Epoch 4 / 15) train acc: 0.296000; val_acc: 0.306000
(Iteration 2001 / 7350) loss: 1.867411
(Iteration 2101 / 7350) loss: 1.924708
(Iteration 2201 / 7350) loss: 1.958967
(Iteration 2301 / 7350) loss: 1.958663
(Iteration 2401 / 7350) loss: 1.667270
(Epoch 5 / 15) train acc: 0.367000; val_acc: 0.369000
(Iteration 2501 / 7350) loss: 1.723268
(Iteration 2601 / 7350) loss: 1.691806
(Iteration 2701 / 7350) loss: 1.752064
(Iteration 2801 / 7350) loss: 1.796572
(Iteration 2901 / 7350) loss: 1.732173
(Epoch 6 / 15) train acc: 0.413000; val_acc: 0.390000
(Iteration 3001 / 7350) loss: 1.599744
(Iteration 3101 / 7350) loss: 1.635335
(Iteration 3201 / 7350) loss: 1.651562
(Iteration 3301 / 7350) loss: 1.613194
(Iteration 3401 / 7350) loss: 1.616407
(Epoch 7 / 15) train acc: 0.418000; val_acc: 0.419000
(Iteration 3501 / 7350) loss: 1.635156
(Iteration 3601 / 7350) loss: 1.588911
(Iteration 3701 / 7350) loss: 1.701666
(Iteration 3801 / 7350) loss: 1.619480
(Iteration 3901 / 7350) loss: 1.605041
(Epoch 8 / 15) train acc: 0.456000; val_acc: 0.444000
(Iteration 4001 / 7350) loss: 1.724950
(Iteration 4101 / 7350) loss: 1.498256
(Iteration 4201 / 7350) loss: 1.426431
(Iteration 4301 / 7350) loss: 1.601792
(Iteration 4401 / 7350) loss: 1.442493
(Epoch 9 / 15) train acc: 0.459000; val_acc: 0.456000
(Iteration 4501 / 7350) loss: 1.599653
(Iteration 4601 / 7350) loss: 1.517882
```

```
(Iteration 4701 / 7350) loss: 1.564454
(Iteration 4801 / 7350) loss: 1.479783
(Epoch 10 / 15) train acc: 0.466000; val_acc: 0.477000
(Iteration 4901 / 7350) loss: 1.557490
(Iteration 5001 / 7350) loss: 1.626906
(Iteration 5101 / 7350) loss: 1.516359
(Iteration 5201 / 7350) loss: 1.389902
(Iteration 5301 / 7350) loss: 1.496768
(Epoch 11 / 15) train acc: 0.519000; val_acc: 0.481000
(Iteration 5401 / 7350) loss: 1.474336
(Iteration 5501 / 7350) loss: 1.532291
(Iteration 5601 / 7350) loss: 1.518013
(Iteration 5701 / 7350) loss: 1.450413
(Iteration 5801 / 7350) loss: 1.341100
(Epoch 12 / 15) train acc: 0.476000; val_acc: 0.481000
(Iteration 5901 / 7350) loss: 1.589466
(Iteration 6001 / 7350) loss: 1.355717
(Iteration 6101 / 7350) loss: 1.605986
(Iteration 6201 / 7350) loss: 1.400818
(Iteration 6301 / 7350) loss: 1.477872
(Epoch 13 / 15) train acc: 0.518000; val_acc: 0.490000
(Iteration 6401 / 7350) loss: 1.636412
(Iteration 6501 / 7350) loss: 1.412547
(Iteration 6601 / 7350) loss: 1.469488
(Iteration 6701 / 7350) loss: 1.409900
(Iteration 6801 / 7350) loss: 1.547290
(Epoch 14 / 15) train acc: 0.539000; val_acc: 0.499000
(Iteration 6901 / 7350) loss: 1.287994
(Iteration 7001 / 7350) loss: 1.454485
(Iteration 7101 / 7350) loss: 1.465108
(Iteration 7201 / 7350) loss: 1.553518
(Iteration 7301 / 7350) loss: 1.337121
(Epoch 15 / 15) train acc: 0.509000; val_acc: 0.500000
Best model not updated! Current best_val_acc = 0.606
Start computing trial 26/30...
reg: 0.0013357319512817655, lr: 0.015563750624970042, lr_decay:
0.9783186637076792
(Iteration 1 / 490) loss: 2.302623
(Epoch 0 / 1) train acc: 0.104000; val_acc: 0.082000
(Iteration 101 / 490) loss: 2.302531
(Iteration 201 / 490) loss: 2.301148
(Iteration 301 / 490) loss: 2.299887
(Iteration 401 / 490) loss: 2.296539
(Epoch 1 / 1) train acc: 0.245000; val_acc: 0.271000
Start computing trial 26/30...
reg: 0.0013357319512817655, lr: 0.015563750624970042, lr_decay:
0.9783186637076792
(Iteration 1 / 7350) loss: 2.302637
```

```
(Epoch 0 / 15) train acc: 0.101000; val_acc: 0.110000
(Iteration 101 / 7350) loss: 2.302615
(Iteration 201 / 7350) loss: 2.302178
(Iteration 301 / 7350) loss: 2.299820
(Iteration 401 / 7350) loss: 2.297194
(Epoch 1 / 15) train acc: 0.246000; val_acc: 0.236000
(Iteration 501 / 7350) loss: 2.280912
(Iteration 601 / 7350) loss: 2.263244
(Iteration 701 / 7350) loss: 2.238466
(Iteration 801 / 7350) loss: 2.101144
(Iteration 901 / 7350) loss: 2.063520
(Epoch 2 / 15) train acc: 0.281000; val_acc: 0.305000
(Iteration 1001 / 7350) loss: 1.903454
(Iteration 1101 / 7350) loss: 1.955800
(Iteration 1201 / 7350) loss: 1.802781
(Iteration 1301 / 7350) loss: 1.946009
(Iteration 1401 / 7350) loss: 1.765384
(Epoch 3 / 15) train acc: 0.383000; val_acc: 0.389000
(Iteration 1501 / 7350) loss: 1.677079
(Iteration 1601 / 7350) loss: 1.635203
(Iteration 1701 / 7350) loss: 1.697866
(Iteration 1801 / 7350) loss: 1.591764
(Iteration 1901 / 7350) loss: 1.487969
(Epoch 4 / 15) train acc: 0.447000; val_acc: 0.427000
(Iteration 2001 / 7350) loss: 1.619100
(Iteration 2101 / 7350) loss: 1.448919
(Iteration 2201 / 7350) loss: 1.532235
(Iteration 2301 / 7350) loss: 1.635189
(Iteration 2401 / 7350) loss: 1.471607
(Epoch 5 / 15) train acc: 0.499000; val_acc: 0.465000
(Iteration 2501 / 7350) loss: 1.550099
(Iteration 2601 / 7350) loss: 1.409579
(Iteration 2701 / 7350) loss: 1.357556
(Iteration 2801 / 7350) loss: 1.501666
(Iteration 2901 / 7350) loss: 1.427327
(Epoch 6 / 15) train acc: 0.487000; val_acc: 0.475000
(Iteration 3001 / 7350) loss: 1.495082
(Iteration 3101 / 7350) loss: 1.417502
(Iteration 3201 / 7350) loss: 1.678322
(Iteration 3301 / 7350) loss: 1.423438
(Iteration 3401 / 7350) loss: 1.380998
(Epoch 7 / 15) train acc: 0.475000; val_acc: 0.502000
(Iteration 3501 / 7350) loss: 1.454138
(Iteration 3601 / 7350) loss: 1.512051
(Iteration 3701 / 7350) loss: 1.322370
(Iteration 3801 / 7350) loss: 1.356652
(Iteration 3901 / 7350) loss: 1.360741
(Epoch 8 / 15) train acc: 0.538000; val_acc: 0.512000
```

```
(Iteration 4001 / 7350) loss: 1.499415
(Iteration 4101 / 7350) loss: 1.389495
(Iteration 4201 / 7350) loss: 1.477840
(Iteration 4301 / 7350) loss: 1.334080
(Iteration 4401 / 7350) loss: 1.264977
(Epoch 9 / 15) train acc: 0.504000; val_acc: 0.512000
(Iteration 4501 / 7350) loss: 1.264054
(Iteration 4601 / 7350) loss: 1.471281
(Iteration 4701 / 7350) loss: 1.369268
(Iteration 4801 / 7350) loss: 1.364627
(Epoch 10 / 15) train acc: 0.516000; val_acc: 0.511000
(Iteration 4901 / 7350) loss: 1.267774
(Iteration 5001 / 7350) loss: 1.402409
(Iteration 5101 / 7350) loss: 1.266872
(Iteration 5201 / 7350) loss: 1.494665
(Iteration 5301 / 7350) loss: 1.395425
(Epoch 11 / 15) train acc: 0.557000; val_acc: 0.513000
(Iteration 5401 / 7350) loss: 1.258674
(Iteration 5501 / 7350) loss: 1.367705
(Iteration 5601 / 7350) loss: 1.421702
(Iteration 5701 / 7350) loss: 1.451995
(Iteration 5801 / 7350) loss: 1.254909
(Epoch 12 / 15) train acc: 0.515000; val_acc: 0.513000
(Iteration 5901 / 7350) loss: 1.300883
(Iteration 6001 / 7350) loss: 1.515767
(Iteration 6101 / 7350) loss: 1.383106
(Iteration 6201 / 7350) loss: 1.098243
(Iteration 6301 / 7350) loss: 1.287745
(Epoch 13 / 15) train acc: 0.514000; val_acc: 0.514000
(Iteration 6401 / 7350) loss: 1.380655
(Iteration 6501 / 7350) loss: 1.406634
(Iteration 6601 / 7350) loss: 1.369640
(Iteration 6701 / 7350) loss: 1.327401
(Iteration 6801 / 7350) loss: 1.382203
(Epoch 14 / 15) train acc: 0.546000; val_acc: 0.522000
(Iteration 6901 / 7350) loss: 1.377081
(Iteration 7001 / 7350) loss: 1.142854
(Iteration 7101 / 7350) loss: 1.395317
(Iteration 7201 / 7350) loss: 1.184615
(Iteration 7301 / 7350) loss: 1.267892
(Epoch 15 / 15) train acc: 0.573000; val_acc: 0.519000
Best model not updated! Current best_val_acc = 0.606
Start computing trial 27/30...
reg: 0.008123369889001903, lr: 0.0661833910476517, lr_decay: 0.9834350731415992
(Iteration 1 / 490) loss: 2.302927
(Epoch 0 / 1) train acc: 0.105000; val_acc: 0.103000
(Iteration 101 / 490) loss: 2.295674
(Iteration 201 / 490) loss: 2.038312
```

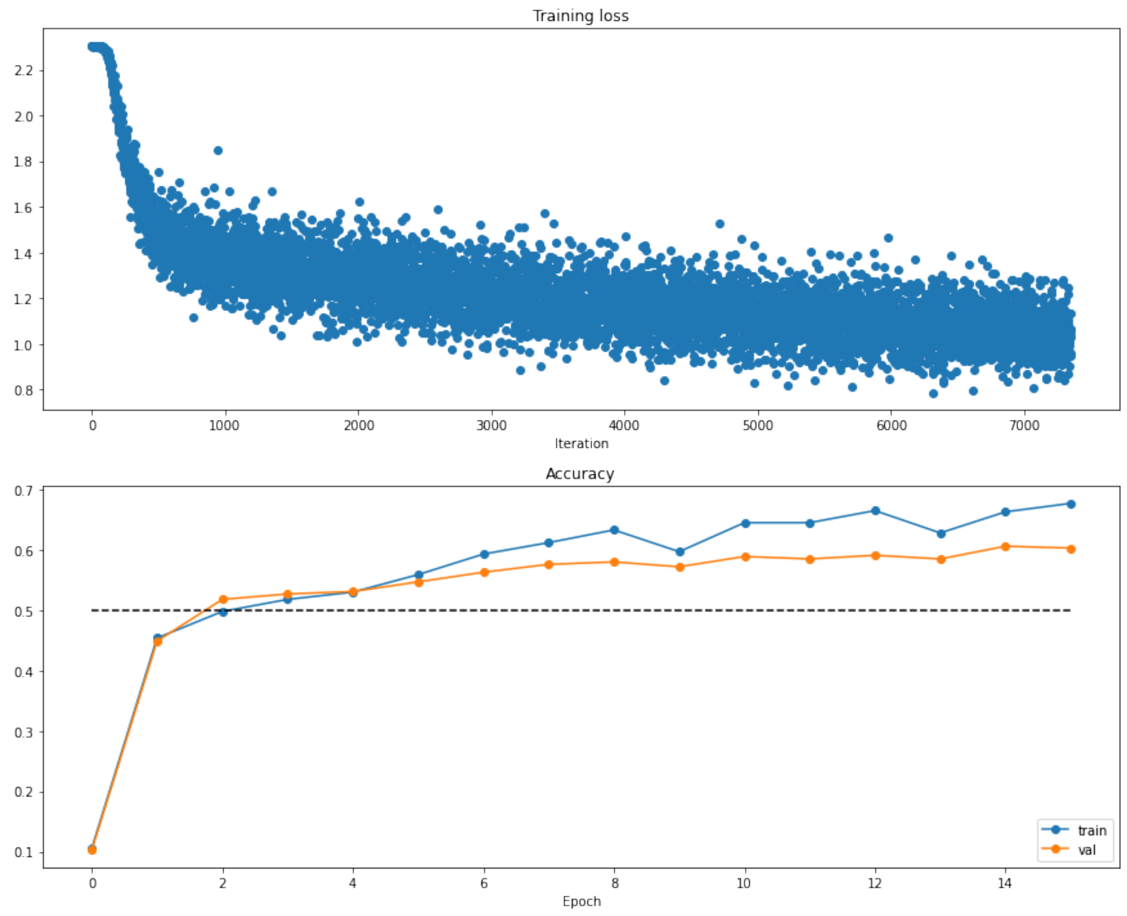
```
(Iteration 301 / 490) loss: 1.892966
(Iteration 401 / 490) loss: 1.719499
(Epoch 1 / 1) train acc: 0.455000; val_acc: 0.437000
Start computing trial 27/30...
reg: 0.008123369889001903, lr: 0.0661833910476517, lr_decay: 0.9834350731415992
(Iteration 1 / 7350) loss: 2.302877
(Epoch 0 / 15) train acc: 0.086000; val_acc: 0.087000
(Iteration 101 / 7350) loss: 2.299871
(Iteration 201 / 7350) loss: 2.038095
(Iteration 301 / 7350) loss: 1.786320
(Iteration 401 / 7350) loss: 1.720959
(Epoch 1 / 15) train acc: 0.464000; val_acc: 0.436000
(Iteration 501 / 7350) loss: 1.545910
(Iteration 601 / 7350) loss: 1.621691
(Iteration 701 / 7350) loss: 1.435915
(Iteration 801 / 7350) loss: 1.218202
(Iteration 901 / 7350) loss: 1.430972
(Epoch 2 / 15) train acc: 0.518000; val_acc: 0.491000
(Iteration 1001 / 7350) loss: 1.518646
(Iteration 1101 / 7350) loss: 1.626938
(Iteration 1201 / 7350) loss: 1.476875
(Iteration 1301 / 7350) loss: 1.502012
(Iteration 1401 / 7350) loss: 1.475000
(Epoch 3 / 15) train acc: 0.513000; val_acc: 0.504000
(Iteration 1501 / 7350) loss: 1.566237
(Iteration 1601 / 7350) loss: 1.420812
(Iteration 1701 / 7350) loss: 1.521905
(Iteration 1801 / 7350) loss: 1.233894
(Iteration 1901 / 7350) loss: 1.476790
(Epoch 4 / 15) train acc: 0.525000; val_acc: 0.517000
(Iteration 2001 / 7350) loss: 1.394324
(Iteration 2101 / 7350) loss: 1.268920
(Iteration 2201 / 7350) loss: 1.320551
(Iteration 2301 / 7350) loss: 1.336105
(Iteration 2401 / 7350) loss: 1.390688
(Epoch 5 / 15) train acc: 0.535000; val_acc: 0.539000
(Iteration 2501 / 7350) loss: 1.379880
(Iteration 2601 / 7350) loss: 1.468098
(Iteration 2701 / 7350) loss: 1.395458
(Iteration 2801 / 7350) loss: 1.700214
(Iteration 2901 / 7350) loss: 1.356894
(Epoch 6 / 15) train acc: 0.562000; val_acc: 0.528000
(Iteration 3001 / 7350) loss: 1.343881
(Iteration 3101 / 7350) loss: 1.389019
(Iteration 3201 / 7350) loss: 1.482927
(Iteration 3301 / 7350) loss: 1.428818
(Iteration 3401 / 7350) loss: 1.326978
(Epoch 7 / 15) train acc: 0.520000; val_acc: 0.551000
```

```
(Iteration 3501 / 7350) loss: 1.415149
(Iteration 3601 / 7350) loss: 1.294358
(Iteration 3701 / 7350) loss: 1.419543
(Iteration 3801 / 7350) loss: 1.475140
(Iteration 3901 / 7350) loss: 1.460139
(Epoch 8 / 15) train acc: 0.560000; val_acc: 0.528000
(Iteration 4001 / 7350) loss: 1.389644
(Iteration 4101 / 7350) loss: 1.435404
(Iteration 4201 / 7350) loss: 1.503903
(Iteration 4301 / 7350) loss: 1.357463
(Iteration 4401 / 7350) loss: 1.323429
(Epoch 9 / 15) train acc: 0.597000; val_acc: 0.552000
(Iteration 4501 / 7350) loss: 1.485123
(Iteration 4601 / 7350) loss: 1.296751
(Iteration 4701 / 7350) loss: 1.364040
(Iteration 4801 / 7350) loss: 1.386246
(Epoch 10 / 15) train acc: 0.589000; val_acc: 0.554000
(Iteration 4901 / 7350) loss: 1.373188
(Iteration 5001 / 7350) loss: 1.165306
(Iteration 5101 / 7350) loss: 1.363133
(Iteration 5201 / 7350) loss: 1.402324
(Iteration 5301 / 7350) loss: 1.467330
(Epoch 11 / 15) train acc: 0.593000; val_acc: 0.554000
(Iteration 5401 / 7350) loss: 1.349414
(Iteration 5501 / 7350) loss: 1.354070
(Iteration 5601 / 7350) loss: 1.416528
(Iteration 5701 / 7350) loss: 1.355616
(Iteration 5801 / 7350) loss: 1.551467
(Epoch 12 / 15) train acc: 0.557000; val_acc: 0.553000
(Iteration 5901 / 7350) loss: 1.386129
(Iteration 6001 / 7350) loss: 1.421546
(Iteration 6101 / 7350) loss: 1.351365
(Iteration 6201 / 7350) loss: 1.388513
(Iteration 6301 / 7350) loss: 1.511163
(Epoch 13 / 15) train acc: 0.598000; val_acc: 0.548000
(Iteration 6401 / 7350) loss: 1.327662
(Iteration 6501 / 7350) loss: 1.362819
(Iteration 6601 / 7350) loss: 1.245752
(Iteration 6701 / 7350) loss: 1.360500
(Iteration 6801 / 7350) loss: 1.402114
(Epoch 14 / 15) train acc: 0.611000; val_acc: 0.551000
(Iteration 6901 / 7350) loss: 1.412846
(Iteration 7001 / 7350) loss: 1.437754
(Iteration 7101 / 7350) loss: 1.411069
(Iteration 7201 / 7350) loss: 1.196405
(Iteration 7301 / 7350) loss: 1.297998
(Epoch 15 / 15) train acc: 0.606000; val_acc: 0.562000
Best model not updated! Current best_val_acc = 0.606
```

```
Start computing trial 28/30...
reg: 0.0026336863836840542, lr: 0.05576911875745303, lr_decay:
0.9804476525176241
(Iteration 1 / 490) loss: 2.302684
(Epoch 0 / 1) train acc: 0.084000; val_acc: 0.102000
(Iteration 101 / 490) loss: 2.299071
(Iteration 201 / 490) loss: 2.215689
(Iteration 301 / 490) loss: 1.914192
(Iteration 401 / 490) loss: 1.655165
(Epoch 1 / 1) train acc: 0.429000; val_acc: 0.412000
Start computing trial 28/30...
reg: 0.0026336863836840542, lr: 0.05576911875745303, lr_decay:
0.9804476525176241
(Iteration 1 / 7350) loss: 2.302728
(Epoch 0 / 15) train acc: 0.079000; val_acc: 0.087000
(Iteration 101 / 7350) loss: 2.297440
(Iteration 201 / 7350) loss: 2.202260
(Iteration 301 / 7350) loss: 1.899292
(Iteration 401 / 7350) loss: 1.782875
(Epoch 1 / 15) train acc: 0.436000; val_acc: 0.422000
(Iteration 501 / 7350) loss: 1.644341
(Iteration 601 / 7350) loss: 1.603679
(Iteration 701 / 7350) loss: 1.472912
(Iteration 801 / 7350) loss: 1.462956
(Iteration 901 / 7350) loss: 1.367733
(Epoch 2 / 15) train acc: 0.497000; val_acc: 0.507000
(Iteration 1001 / 7350) loss: 1.468453
(Iteration 1101 / 7350) loss: 1.342590
(Iteration 1201 / 7350) loss: 1.516681
(Iteration 1301 / 7350) loss: 1.335180
(Iteration 1401 / 7350) loss: 1.277701
(Epoch 3 / 15) train acc: 0.522000; val_acc: 0.520000
(Iteration 1501 / 7350) loss: 1.379810
(Iteration 1601 / 7350) loss: 1.466399
(Iteration 1701 / 7350) loss: 1.456947
(Iteration 1801 / 7350) loss: 1.371668
(Iteration 1901 / 7350) loss: 1.288519
(Epoch 4 / 15) train acc: 0.545000; val_acc: 0.507000
(Iteration 2001 / 7350) loss: 1.235394
(Iteration 2101 / 7350) loss: 1.388049
(Iteration 2201 / 7350) loss: 1.269885
(Iteration 2301 / 7350) loss: 1.232244
(Iteration 2401 / 7350) loss: 1.330320
(Epoch 5 / 15) train acc: 0.533000; val_acc: 0.521000
(Iteration 2501 / 7350) loss: 1.340553
(Iteration 2601 / 7350) loss: 1.420207
(Iteration 2701 / 7350) loss: 1.180824
(Iteration 2801 / 7350) loss: 1.451068
```

```
(Iteration 2901 / 7350) loss: 1.301308
(Epoch 6 / 15) train acc: 0.560000; val_acc: 0.539000
(Iteration 3001 / 7350) loss: 1.249512
(Iteration 3101 / 7350) loss: 1.374403
(Iteration 3201 / 7350) loss: 1.227124
(Iteration 3301 / 7350) loss: 1.162641
(Iteration 3401 / 7350) loss: 1.533330
(Epoch 7 / 15) train acc: 0.561000; val_acc: 0.547000
(Iteration 3501 / 7350) loss: 1.186521
(Iteration 3601 / 7350) loss: 1.384452
(Iteration 3701 / 7350) loss: 1.140711
(Iteration 3801 / 7350) loss: 1.263319
(Iteration 3901 / 7350) loss: 1.316573
(Epoch 8 / 15) train acc: 0.585000; val_acc: 0.560000
(Iteration 4001 / 7350) loss: 1.267158
(Iteration 4101 / 7350) loss: 1.185143
(Iteration 4201 / 7350) loss: 1.276614
(Iteration 4301 / 7350) loss: 1.252422
(Iteration 4401 / 7350) loss: 1.246425
(Epoch 9 / 15) train acc: 0.587000; val_acc: 0.561000
(Iteration 4501 / 7350) loss: 1.167157
(Iteration 4601 / 7350) loss: 1.136913
(Iteration 4701 / 7350) loss: 1.139030
(Iteration 4801 / 7350) loss: 1.318615
(Epoch 10 / 15) train acc: 0.590000; val_acc: 0.565000
(Iteration 4901 / 7350) loss: 1.097150
(Iteration 5001 / 7350) loss: 1.262947
(Iteration 5101 / 7350) loss: 1.264295
(Iteration 5201 / 7350) loss: 1.278384
(Iteration 5301 / 7350) loss: 1.265250
(Epoch 11 / 15) train acc: 0.607000; val_acc: 0.570000
(Iteration 5401 / 7350) loss: 1.252370
(Iteration 5501 / 7350) loss: 0.977873
(Iteration 5601 / 7350) loss: 1.081838
(Iteration 5701 / 7350) loss: 1.313634
(Iteration 5801 / 7350) loss: 1.114673
(Epoch 12 / 15) train acc: 0.607000; val_acc: 0.571000
(Iteration 5901 / 7350) loss: 1.482647
(Iteration 6001 / 7350) loss: 1.220562
(Iteration 6101 / 7350) loss: 1.388294
(Iteration 6201 / 7350) loss: 1.014699
(Iteration 6301 / 7350) loss: 1.109285
(Epoch 13 / 15) train acc: 0.613000; val_acc: 0.590000
(Iteration 6401 / 7350) loss: 1.212591
(Iteration 6501 / 7350) loss: 1.174271
(Iteration 6601 / 7350) loss: 1.141943
(Iteration 6701 / 7350) loss: 1.136610
(Iteration 6801 / 7350) loss: 1.341834
```

```
(Epoch 14 / 15) train acc: 0.657000; val_acc: 0.577000
(Iteration 6901 / 7350) loss: 1.175852
(Iteration 7001 / 7350) loss: 1.140276
(Iteration 7101 / 7350) loss: 1.077515
(Iteration 7201 / 7350) loss: 1.223817
(Iteration 7301 / 7350) loss: 1.203876
(Epoch 15 / 15) train acc: 0.583000; val_acc: 0.586000
Best model not updated! Current best_val_acc = 0.606
Start computing trial 29/30...
reg: 0.004479696054340802, lr: 0.012687472592695778, lr_decay:
0.9808054937260614
(Iteration 1 / 490) loss: 2.302777
(Epoch 0 / 1) train acc: 0.111000; val_acc: 0.082000
(Iteration 101 / 490) loss: 2.302572
(Iteration 201 / 490) loss: 2.301892
(Iteration 301 / 490) loss: 2.302313
(Iteration 401 / 490) loss: 2.298194
(Epoch 1 / 1) train acc: 0.182000; val_acc: 0.178000
```



[25]:
```python
# Run your best neural net classifier on the test set. You should be able
# to get more than 55% accuracy.

y_test_pred = np.argmax(best_net.loss(data['X_test']), axis=1)
test_acc = (y_test_pred == data['y_test']).mean()
print(test_acc)
```

```
0.597
```