two_layer_net

February 1, 2024

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

# TODO: Enter the foldername in your Drive where you have saved the unzipped
# assignment folder, e.g. 'cse493g1/assignments/assignment2/'
FOLDERNAME = 'cse493g1/assignments/assignment2/'
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 /content/drive/My\ Drive/$FOLDERNAME/cse493g1/datasets/
!bash get_datasets.sh
%cd /content/drive/My\ Drive/$FOLDERNAME
```

```
Mounted at /content/drive
/content/drive/My Drive/cse493g1/assignments/assignment2/cse493g1/datasets
/content/drive/My Drive/cse493g1/assignments/assignment2
```

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 cse493g1.classifiers.fc_net import *
from cse493g1.data_utils import get_CIFAR10_data
from cse493g1.gradient_check import eval_numerical_gradient,␣
 ↪eval_numerical_gradient_array
from cse493g1.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 `cse493g1/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(493)
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:  2.7815615012337633e-10
dw error:  4.28112143997314e-11
db error:  7.32931845803195e-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(493)
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:  3.275625790132643e-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:

Sigmoid and ReLu also have this problem. For sigmoid, It's the input containing numbers that are very positive or very negative. For ReLu, it's the input containing negative numbers.

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 `cse493g1/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]:
```
from cse493g1.layer_utils import affine_relu_forward, affine_relu_backward
np.random.seed(493)
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:  4.791696167224978e-10
dw error:  1.877217636859801e-10
db error:  8.009352854936581e-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 `cse493g1/layers.py`. These should be similar to what you implemented in `cse493g1/classifiers/softmax.py` and `cse493g1/classifiers/linear_svm.py` in Assignment 1.

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

[10]:
```
np.random.seed(493)
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
print('\nTesting softmax_loss:')
print('loss: ', loss)
print('dx error: ', rel_error(dx_num, dx))
```

```
Testing svm_loss:
loss:  8.998158338429793
dx error:  8.182894472887002e-10

Testing softmax_loss:
loss:  2.3024013710141706
dx error:  7.452229549289443e-09
```

8 Two-layer network

Open the file `cse493g1/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(493)
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: 3.31e-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: 2.85e-08
b1 relative error: 1.56e-08
b2 relative error: 7.76e-10
```

9 Solver

Open the file `cse493g1/solver.py` and read through it to familiarize yourself with the API. Additionally, familiarize yourself with the `sgd` function in `cse493g1/optim.py`. After doing so, use a `Solver` instance to train a `TwoLayerNet` that achieves about `36%` accuracy on the validation set.

[19]:
```
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-4,
                    },
                    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: 2.303174
(Epoch 0 / 10) train acc: 0.082000; val_acc: 0.095000
(Iteration 101 / 4900) loss: 2.263886
(Iteration 201 / 4900) loss: 2.180113
(Iteration 301 / 4900) loss: 1.978916
(Iteration 401 / 4900) loss: 1.955108
(Epoch 1 / 10) train acc: 0.290000; val_acc: 0.300000
(Iteration 501 / 4900) loss: 1.955945
```

```
(Iteration 601 / 4900) loss: 1.991997
(Iteration 701 / 4900) loss: 1.850502
(Iteration 801 / 4900) loss: 1.703030
(Iteration 901 / 4900) loss: 1.884382
(Epoch 2 / 10) train acc: 0.355000; val_acc: 0.363000
(Iteration 1001 / 4900) loss: 1.774429
(Iteration 1101 / 4900) loss: 1.745825
(Iteration 1201 / 4900) loss: 1.647701
(Iteration 1301 / 4900) loss: 1.880994
(Iteration 1401 / 4900) loss: 1.835798
(Epoch 3 / 10) train acc: 0.398000; val_acc: 0.396000
(Iteration 1501 / 4900) loss: 1.806782
(Iteration 1601 / 4900) loss: 1.720693
(Iteration 1701 / 4900) loss: 1.740815
(Iteration 1801 / 4900) loss: 1.698166
(Iteration 1901 / 4900) loss: 1.595304
(Epoch 4 / 10) train acc: 0.402000; val_acc: 0.416000
(Iteration 2001 / 4900) loss: 1.661073
(Iteration 2101 / 4900) loss: 1.596802
(Iteration 2201 / 4900) loss: 1.694312
(Iteration 2301 / 4900) loss: 1.740139
(Iteration 2401 / 4900) loss: 1.705379
(Epoch 5 / 10) train acc: 0.431000; val_acc: 0.435000
(Iteration 2501 / 4900) loss: 1.830258
(Iteration 2601 / 4900) loss: 1.544960
(Iteration 2701 / 4900) loss: 1.481980
(Iteration 2801 / 4900) loss: 1.662649
(Iteration 2901 / 4900) loss: 1.492310
(Epoch 6 / 10) train acc: 0.438000; val_acc: 0.441000
(Iteration 3001 / 4900) loss: 1.576741
(Iteration 3101 / 4900) loss: 1.568374
(Iteration 3201 / 4900) loss: 1.722623
(Iteration 3301 / 4900) loss: 1.606164
(Iteration 3401 / 4900) loss: 1.701587
(Epoch 7 / 10) train acc: 0.459000; val_acc: 0.450000
(Iteration 3501 / 4900) loss: 1.614845
(Iteration 3601 / 4900) loss: 1.580161
(Iteration 3701 / 4900) loss: 1.745413
(Iteration 3801 / 4900) loss: 1.652085
(Iteration 3901 / 4900) loss: 1.322795
(Epoch 8 / 10) train acc: 0.452000; val_acc: 0.456000
(Iteration 4001 / 4900) loss: 1.640874
(Iteration 4101 / 4900) loss: 1.429097
(Iteration 4201 / 4900) loss: 1.457338
(Iteration 4301 / 4900) loss: 1.513188
(Iteration 4401 / 4900) loss: 1.485017
(Epoch 9 / 10) train acc: 0.470000; val_acc: 0.458000
(Iteration 4501 / 4900) loss: 1.607762
```

```
(Iteration 4601 / 4900) loss: 1.573608
(Iteration 4701 / 4900) loss: 1.524841
(Iteration 4801 / 4900) loss: 1.574602
(Epoch 10 / 10) train acc: 0.458000; val_acc: 0.464000
```

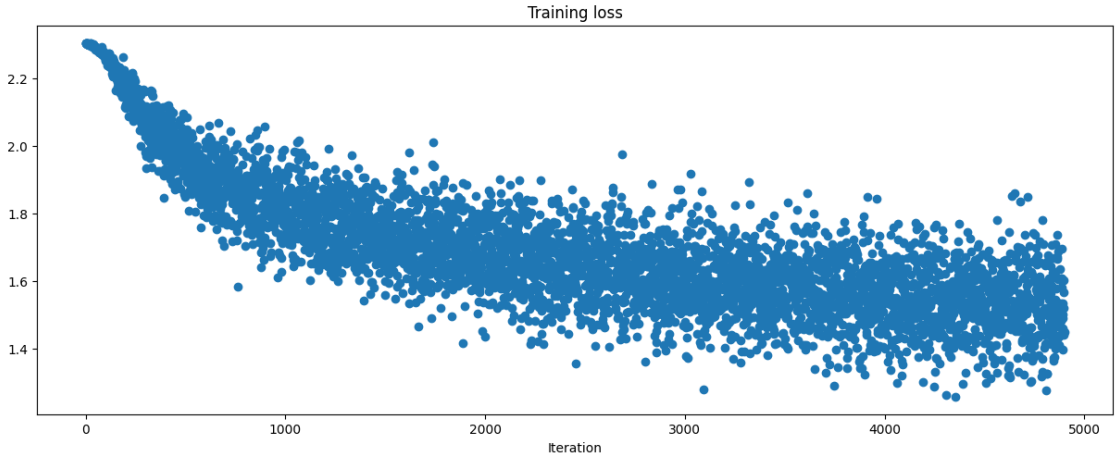
10 Debug the training

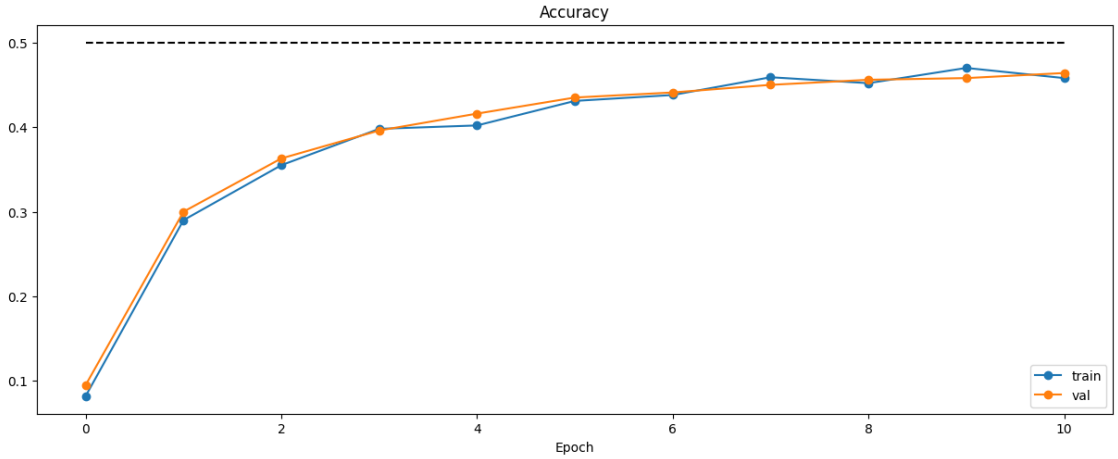
With the default parameters we provided above, you should get a validation accuracy of about 0.36 on the validation set. This isn't very good.

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.

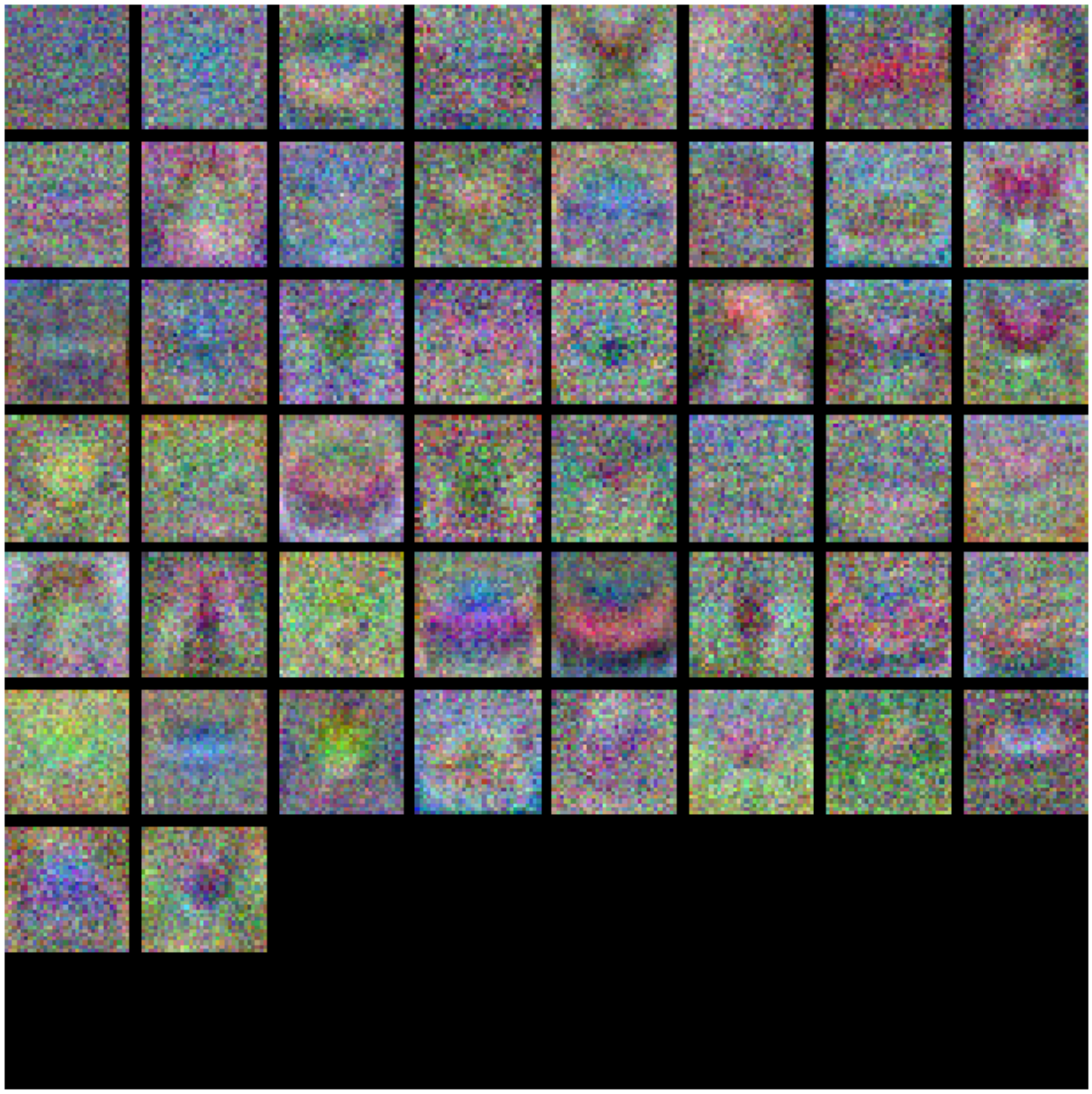
Another strategy is to visualize the weights that were learned in the first layer of the network. In most neural networks trained on visual data, the first layer weights typically show some visible structure when visualized.

[20]:
```
# 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()
```





[21]:
```python
from cse493g1.vis_utils import visualize_grid

# Visualize the weights of the network

def show_net_weights(net):
    W1 = net.params['W1']
    W1 = W1.reshape(3, 32, 32, -1).transpose(3, 1, 2, 0)
    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.).

[26]:
```
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)*****

learning_rates = [1e-3, 5e-4, 1e-4]
hidden_size = [60, 70, 80]
best_acc = -1

for lr in learning_rates:
  for size in hidden_size:
    model = TwoLayerNet(input_size, size, num_classes)

    solver = Solver(model, data,
                    update_rule='sgd',
```

```
                        optim_config={
                          'learning_rate': lr,
                        },
                        lr_decay=0.8,
                        num_epochs=10, batch_size=100,
                        print_every=100)
    solver.train()
    if solver.best_val_acc>best_acc:
      best_acc = solver.best_val_acc
      best_model = model

print('test accuracy: ', solver.best_val_acc)


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

```
(Iteration 1 / 4900) loss: 2.304370
(Epoch 0 / 10) train acc: 0.126000; val_acc: 0.121000
(Iteration 101 / 4900) loss: 2.008099
(Iteration 201 / 4900) loss: 1.627927
(Iteration 301 / 4900) loss: 1.600405
(Iteration 401 / 4900) loss: 1.658272
(Epoch 1 / 10) train acc: 0.442000; val_acc: 0.426000
(Iteration 501 / 4900) loss: 1.416244
(Iteration 601 / 4900) loss: 1.662964
(Iteration 701 / 4900) loss: 1.671242
(Iteration 801 / 4900) loss: 1.554091
(Iteration 901 / 4900) loss: 1.439983
(Epoch 2 / 10) train acc: 0.465000; val_acc: 0.479000
(Iteration 1001 / 4900) loss: 1.457831
(Iteration 1101 / 4900) loss: 1.436472
(Iteration 1201 / 4900) loss: 1.433358
(Iteration 1301 / 4900) loss: 1.335860
(Iteration 1401 / 4900) loss: 1.397290
(Epoch 3 / 10) train acc: 0.522000; val_acc: 0.479000
(Iteration 1501 / 4900) loss: 1.336865
(Iteration 1601 / 4900) loss: 1.311495
(Iteration 1701 / 4900) loss: 1.449863
(Iteration 1801 / 4900) loss: 1.383777
(Iteration 1901 / 4900) loss: 1.372879
(Epoch 4 / 10) train acc: 0.509000; val_acc: 0.503000
(Iteration 2001 / 4900) loss: 1.312452
(Iteration 2101 / 4900) loss: 1.596388
(Iteration 2201 / 4900) loss: 1.367794
```

```
(Iteration 2301 / 4900) loss: 1.460798
(Iteration 2401 / 4900) loss: 1.386785
(Epoch 5 / 10) train acc: 0.525000; val_acc: 0.503000
(Iteration 2501 / 4900) loss: 1.289509
(Iteration 2601 / 4900) loss: 1.161424
(Iteration 2701 / 4900) loss: 1.182718
(Iteration 2801 / 4900) loss: 1.201102
(Iteration 2901 / 4900) loss: 1.540045
(Epoch 6 / 10) train acc: 0.527000; val_acc: 0.519000
(Iteration 3001 / 4900) loss: 1.294167
(Iteration 3101 / 4900) loss: 1.282920
(Iteration 3201 / 4900) loss: 1.344436
(Iteration 3301 / 4900) loss: 1.269042
(Iteration 3401 / 4900) loss: 1.170739
(Epoch 7 / 10) train acc: 0.562000; val_acc: 0.512000
(Iteration 3501 / 4900) loss: 1.317591
(Iteration 3601 / 4900) loss: 1.215471
(Iteration 3701 / 4900) loss: 1.122689
(Iteration 3801 / 4900) loss: 1.324620
(Iteration 3901 / 4900) loss: 1.149814
(Epoch 8 / 10) train acc: 0.582000; val_acc: 0.509000
(Iteration 4001 / 4900) loss: 1.161186
(Iteration 4101 / 4900) loss: 1.092314
(Iteration 4201 / 4900) loss: 1.315460
(Iteration 4301 / 4900) loss: 1.262869
(Iteration 4401 / 4900) loss: 1.418042
(Epoch 9 / 10) train acc: 0.551000; val_acc: 0.500000
(Iteration 4501 / 4900) loss: 1.252901
(Iteration 4601 / 4900) loss: 1.292947
(Iteration 4701 / 4900) loss: 1.333114
(Iteration 4801 / 4900) loss: 1.157275
(Epoch 10 / 10) train acc: 0.582000; val_acc: 0.514000
(Iteration 1 / 4900) loss: 2.305697
(Epoch 0 / 10) train acc: 0.104000; val_acc: 0.106000
(Iteration 101 / 4900) loss: 1.738772
(Iteration 201 / 4900) loss: 1.720488
(Iteration 301 / 4900) loss: 1.698737
(Iteration 401 / 4900) loss: 1.594803
(Epoch 1 / 10) train acc: 0.456000; val_acc: 0.458000
(Iteration 501 / 4900) loss: 1.606652
(Iteration 601 / 4900) loss: 1.550959
(Iteration 701 / 4900) loss: 1.366816
(Iteration 801 / 4900) loss: 1.408726
(Iteration 901 / 4900) loss: 1.548983
(Epoch 2 / 10) train acc: 0.499000; val_acc: 0.460000
(Iteration 1001 / 4900) loss: 1.280085
(Iteration 1101 / 4900) loss: 1.144045
(Iteration 1201 / 4900) loss: 1.419764
```

```
(Iteration 1301 / 4900) loss: 1.492258
(Iteration 1401 / 4900) loss: 1.340604
(Epoch 3 / 10) train acc: 0.513000; val_acc: 0.467000
(Iteration 1501 / 4900) loss: 1.348846
(Iteration 1601 / 4900) loss: 1.397917
(Iteration 1701 / 4900) loss: 1.470231
(Iteration 1801 / 4900) loss: 1.475421
(Iteration 1901 / 4900) loss: 1.410562
(Epoch 4 / 10) train acc: 0.526000; val_acc: 0.501000
(Iteration 2001 / 4900) loss: 1.687188
(Iteration 2101 / 4900) loss: 1.434789
(Iteration 2201 / 4900) loss: 1.534220
(Iteration 2301 / 4900) loss: 1.222167
(Iteration 2401 / 4900) loss: 1.275673
(Epoch 5 / 10) train acc: 0.579000; val_acc: 0.487000
(Iteration 2501 / 4900) loss: 1.331839
(Iteration 2601 / 4900) loss: 1.270413
(Iteration 2701 / 4900) loss: 1.393849
(Iteration 2801 / 4900) loss: 1.379616
(Iteration 2901 / 4900) loss: 1.232057
(Epoch 6 / 10) train acc: 0.540000; val_acc: 0.478000
(Iteration 3001 / 4900) loss: 1.196365
(Iteration 3101 / 4900) loss: 1.224588
(Iteration 3201 / 4900) loss: 1.305239
(Iteration 3301 / 4900) loss: 1.300573
(Iteration 3401 / 4900) loss: 1.387463
(Epoch 7 / 10) train acc: 0.558000; val_acc: 0.498000
(Iteration 3501 / 4900) loss: 1.119219
(Iteration 3601 / 4900) loss: 1.134786
(Iteration 3701 / 4900) loss: 1.145218
(Iteration 3801 / 4900) loss: 1.140625
(Iteration 3901 / 4900) loss: 1.328447
(Epoch 8 / 10) train acc: 0.585000; val_acc: 0.502000
(Iteration 4001 / 4900) loss: 1.235794
(Iteration 4101 / 4900) loss: 1.239619
(Iteration 4201 / 4900) loss: 1.318373
(Iteration 4301 / 4900) loss: 1.069683
(Iteration 4401 / 4900) loss: 1.130772
(Epoch 9 / 10) train acc: 0.594000; val_acc: 0.505000
(Iteration 4501 / 4900) loss: 1.164917
(Iteration 4601 / 4900) loss: 1.146542
(Iteration 4701 / 4900) loss: 1.189869
(Iteration 4801 / 4900) loss: 1.044265
(Epoch 10 / 10) train acc: 0.568000; val_acc: 0.486000
(Iteration 1 / 4900) loss: 2.303708
(Epoch 0 / 10) train acc: 0.154000; val_acc: 0.121000
(Iteration 101 / 4900) loss: 1.744779
(Iteration 201 / 4900) loss: 1.604017
```

```
(Iteration 301 / 4900) loss: 1.550715
(Iteration 401 / 4900) loss: 1.675252
(Epoch 1 / 10) train acc: 0.452000; val_acc: 0.424000
(Iteration 501 / 4900) loss: 1.710851
(Iteration 601 / 4900) loss: 1.387714
(Iteration 701 / 4900) loss: 1.579051
(Iteration 801 / 4900) loss: 1.417635
(Iteration 901 / 4900) loss: 1.345156
(Epoch 2 / 10) train acc: 0.469000; val_acc: 0.460000
(Iteration 1001 / 4900) loss: 1.663729
(Iteration 1101 / 4900) loss: 1.350329
(Iteration 1201 / 4900) loss: 1.390822
(Iteration 1301 / 4900) loss: 1.582549
(Iteration 1401 / 4900) loss: 1.280165
(Epoch 3 / 10) train acc: 0.481000; val_acc: 0.472000
(Iteration 1501 / 4900) loss: 1.224617
(Iteration 1601 / 4900) loss: 1.295408
(Iteration 1701 / 4900) loss: 1.120916
(Iteration 1801 / 4900) loss: 1.161315
(Iteration 1901 / 4900) loss: 1.509739
(Epoch 4 / 10) train acc: 0.527000; val_acc: 0.481000
(Iteration 2001 / 4900) loss: 1.412565
(Iteration 2101 / 4900) loss: 1.214596
(Iteration 2201 / 4900) loss: 1.520391
(Iteration 2301 / 4900) loss: 1.270185
(Iteration 2401 / 4900) loss: 1.174002
(Epoch 5 / 10) train acc: 0.583000; val_acc: 0.490000
(Iteration 2501 / 4900) loss: 1.281854
(Iteration 2601 / 4900) loss: 1.386414
(Iteration 2701 / 4900) loss: 1.174275
(Iteration 2801 / 4900) loss: 1.369770
(Iteration 2901 / 4900) loss: 1.371276
(Epoch 6 / 10) train acc: 0.576000; val_acc: 0.510000
(Iteration 3001 / 4900) loss: 1.226001
(Iteration 3101 / 4900) loss: 1.166645
(Iteration 3201 / 4900) loss: 1.219303
(Iteration 3301 / 4900) loss: 1.216622
(Iteration 3401 / 4900) loss: 1.173357
(Epoch 7 / 10) train acc: 0.567000; val_acc: 0.512000
(Iteration 3501 / 4900) loss: 1.377598
(Iteration 3601 / 4900) loss: 1.359773
(Iteration 3701 / 4900) loss: 1.450044
(Iteration 3801 / 4900) loss: 1.248598
(Iteration 3901 / 4900) loss: 1.225746
(Epoch 8 / 10) train acc: 0.580000; val_acc: 0.502000
(Iteration 4001 / 4900) loss: 1.200648
(Iteration 4101 / 4900) loss: 1.170589
(Iteration 4201 / 4900) loss: 1.092869
```

```
(Iteration 4301 / 4900) loss: 1.099575
(Iteration 4401 / 4900) loss: 1.284097
(Epoch 9 / 10) train acc: 0.568000; val_acc: 0.512000
(Iteration 4501 / 4900) loss: 0.963971
(Iteration 4601 / 4900) loss: 1.183433
(Iteration 4701 / 4900) loss: 1.291313
(Iteration 4801 / 4900) loss: 1.283402
(Epoch 10 / 10) train acc: 0.576000; val_acc: 0.508000
(Iteration 1 / 4900) loss: 2.304140
(Epoch 0 / 10) train acc: 0.120000; val_acc: 0.110000
(Iteration 101 / 4900) loss: 2.090205
(Iteration 201 / 4900) loss: 1.871395
(Iteration 301 / 4900) loss: 1.673234
(Iteration 401 / 4900) loss: 1.740414
(Epoch 1 / 10) train acc: 0.424000; val_acc: 0.430000
(Iteration 501 / 4900) loss: 1.567720
(Iteration 601 / 4900) loss: 1.428549
(Iteration 701 / 4900) loss: 1.555843
(Iteration 801 / 4900) loss: 1.591283
(Iteration 901 / 4900) loss: 1.482992
(Epoch 2 / 10) train acc: 0.445000; val_acc: 0.468000
(Iteration 1001 / 4900) loss: 1.599170
(Iteration 1101 / 4900) loss: 1.612612
(Iteration 1201 / 4900) loss: 1.583102
(Iteration 1301 / 4900) loss: 1.312804
(Iteration 1401 / 4900) loss: 1.443163
(Epoch 3 / 10) train acc: 0.491000; val_acc: 0.475000
(Iteration 1501 / 4900) loss: 1.400491
(Iteration 1601 / 4900) loss: 1.623032
(Iteration 1701 / 4900) loss: 1.369918
(Iteration 1801 / 4900) loss: 1.495091
(Iteration 1901 / 4900) loss: 1.292707
(Epoch 4 / 10) train acc: 0.482000; val_acc: 0.484000
(Iteration 2001 / 4900) loss: 1.382956
(Iteration 2101 / 4900) loss: 1.515468
(Iteration 2201 / 4900) loss: 1.462968
(Iteration 2301 / 4900) loss: 1.435273
(Iteration 2401 / 4900) loss: 1.338811
(Epoch 5 / 10) train acc: 0.500000; val_acc: 0.497000
(Iteration 2501 / 4900) loss: 1.458953
(Iteration 2601 / 4900) loss: 1.308246
(Iteration 2701 / 4900) loss: 1.412885
(Iteration 2801 / 4900) loss: 1.513186
(Iteration 2901 / 4900) loss: 1.453865
(Epoch 6 / 10) train acc: 0.538000; val_acc: 0.499000
(Iteration 3001 / 4900) loss: 1.454273
(Iteration 3101 / 4900) loss: 1.102221
(Iteration 3201 / 4900) loss: 1.388568
```

```
(Iteration 3301 / 4900) loss: 1.358039
(Iteration 3401 / 4900) loss: 1.348672
(Epoch 7 / 10) train acc: 0.498000; val_acc: 0.487000
(Iteration 3501 / 4900) loss: 1.284717
(Iteration 3601 / 4900) loss: 1.397456
(Iteration 3701 / 4900) loss: 1.443424
(Iteration 3801 / 4900) loss: 1.135394
(Iteration 3901 / 4900) loss: 1.224711
(Epoch 8 / 10) train acc: 0.528000; val_acc: 0.493000
(Iteration 4001 / 4900) loss: 1.692458
(Iteration 4101 / 4900) loss: 1.353279
(Iteration 4201 / 4900) loss: 1.494797
(Iteration 4301 / 4900) loss: 1.364392
(Iteration 4401 / 4900) loss: 1.408479
(Epoch 9 / 10) train acc: 0.528000; val_acc: 0.494000
(Iteration 4501 / 4900) loss: 1.407873
(Iteration 4601 / 4900) loss: 1.259278
(Iteration 4701 / 4900) loss: 1.157157
(Iteration 4801 / 4900) loss: 1.278886
(Epoch 10 / 10) train acc: 0.564000; val_acc: 0.502000
(Iteration 1 / 4900) loss: 2.302051
(Epoch 0 / 10) train acc: 0.133000; val_acc: 0.121000
(Iteration 101 / 4900) loss: 1.973335
(Iteration 201 / 4900) loss: 1.815726
(Iteration 301 / 4900) loss: 1.747723
(Iteration 401 / 4900) loss: 1.596239
(Epoch 1 / 10) train acc: 0.414000; val_acc: 0.428000
(Iteration 501 / 4900) loss: 1.559850
(Iteration 601 / 4900) loss: 1.557630
(Iteration 701 / 4900) loss: 1.524037
(Iteration 801 / 4900) loss: 1.512620
(Iteration 901 / 4900) loss: 1.537674
(Epoch 2 / 10) train acc: 0.464000; val_acc: 0.453000
(Iteration 1001 / 4900) loss: 1.487354
(Iteration 1101 / 4900) loss: 1.436672
(Iteration 1201 / 4900) loss: 1.452080
(Iteration 1301 / 4900) loss: 1.449725
(Iteration 1401 / 4900) loss: 1.470369
(Epoch 3 / 10) train acc: 0.487000; val_acc: 0.480000
(Iteration 1501 / 4900) loss: 1.358603
(Iteration 1601 / 4900) loss: 1.468858
(Iteration 1701 / 4900) loss: 1.523678
(Iteration 1801 / 4900) loss: 1.413920
(Iteration 1901 / 4900) loss: 1.390016
(Epoch 4 / 10) train acc: 0.522000; val_acc: 0.487000
(Iteration 2001 / 4900) loss: 1.613648
(Iteration 2101 / 4900) loss: 1.513987
(Iteration 2201 / 4900) loss: 1.407027
```

```
(Iteration 2301 / 4900) loss: 1.335914
(Iteration 2401 / 4900) loss: 1.353355
(Epoch 5 / 10) train acc: 0.530000; val_acc: 0.493000
(Iteration 2501 / 4900) loss: 1.415096
(Iteration 2601 / 4900) loss: 1.313696
(Iteration 2701 / 4900) loss: 1.402551
(Iteration 2801 / 4900) loss: 1.275381
(Iteration 2901 / 4900) loss: 1.329285
(Epoch 6 / 10) train acc: 0.508000; val_acc: 0.493000
(Iteration 3001 / 4900) loss: 1.233353
(Iteration 3101 / 4900) loss: 1.297851
(Iteration 3201 / 4900) loss: 1.459604
(Iteration 3301 / 4900) loss: 1.399788
(Iteration 3401 / 4900) loss: 1.392011
(Epoch 7 / 10) train acc: 0.553000; val_acc: 0.508000
(Iteration 3501 / 4900) loss: 1.403099
(Iteration 3601 / 4900) loss: 1.297057
(Iteration 3701 / 4900) loss: 1.259820
(Iteration 3801 / 4900) loss: 1.297544
(Iteration 3901 / 4900) loss: 1.326631
(Epoch 8 / 10) train acc: 0.539000; val_acc: 0.505000
(Iteration 4001 / 4900) loss: 1.298623
(Iteration 4101 / 4900) loss: 1.235483
(Iteration 4201 / 4900) loss: 1.166861
(Iteration 4301 / 4900) loss: 1.216701
(Iteration 4401 / 4900) loss: 1.359985
(Epoch 9 / 10) train acc: 0.536000; val_acc: 0.512000
(Iteration 4501 / 4900) loss: 1.261680
(Iteration 4601 / 4900) loss: 1.158309
(Iteration 4701 / 4900) loss: 1.471488
(Iteration 4801 / 4900) loss: 1.363091
(Epoch 10 / 10) train acc: 0.542000; val_acc: 0.508000
(Iteration 1 / 4900) loss: 2.309618
(Epoch 0 / 10) train acc: 0.100000; val_acc: 0.112000
(Iteration 101 / 4900) loss: 1.926438
(Iteration 201 / 4900) loss: 1.813570
(Iteration 301 / 4900) loss: 1.795371
(Iteration 401 / 4900) loss: 1.791449
(Epoch 1 / 10) train acc: 0.420000; val_acc: 0.424000
(Iteration 501 / 4900) loss: 1.566694
(Iteration 601 / 4900) loss: 1.680461
(Iteration 701 / 4900) loss: 1.486639
(Iteration 801 / 4900) loss: 1.452523
(Iteration 901 / 4900) loss: 1.601207
(Epoch 2 / 10) train acc: 0.463000; val_acc: 0.460000
(Iteration 1001 / 4900) loss: 1.685691
(Iteration 1101 / 4900) loss: 1.406648
(Iteration 1201 / 4900) loss: 1.355221
```

```
(Iteration 1301 / 4900) loss: 1.445369
(Iteration 1401 / 4900) loss: 1.449505
(Epoch 3 / 10) train acc: 0.491000; val_acc: 0.478000
(Iteration 1501 / 4900) loss: 1.438163
(Iteration 1601 / 4900) loss: 1.433603
(Iteration 1701 / 4900) loss: 1.397552
(Iteration 1801 / 4900) loss: 1.423096
(Iteration 1901 / 4900) loss: 1.327696
(Epoch 4 / 10) train acc: 0.487000; val_acc: 0.495000
(Iteration 2001 / 4900) loss: 1.298788
(Iteration 2101 / 4900) loss: 1.344248
(Iteration 2201 / 4900) loss: 1.320640
(Iteration 2301 / 4900) loss: 1.278839
(Iteration 2401 / 4900) loss: 1.201638
(Epoch 5 / 10) train acc: 0.521000; val_acc: 0.482000
(Iteration 2501 / 4900) loss: 1.449752
(Iteration 2601 / 4900) loss: 1.340283
(Iteration 2701 / 4900) loss: 1.448593
(Iteration 2801 / 4900) loss: 1.385907
(Iteration 2901 / 4900) loss: 1.434101
(Epoch 6 / 10) train acc: 0.542000; val_acc: 0.494000
(Iteration 3001 / 4900) loss: 1.378193
(Iteration 3101 / 4900) loss: 1.445916
(Iteration 3201 / 4900) loss: 1.428992
(Iteration 3301 / 4900) loss: 1.364423
(Iteration 3401 / 4900) loss: 1.374874
(Epoch 7 / 10) train acc: 0.512000; val_acc: 0.509000
(Iteration 3501 / 4900) loss: 1.460170
(Iteration 3601 / 4900) loss: 1.523086
(Iteration 3701 / 4900) loss: 1.371249
(Iteration 3801 / 4900) loss: 1.544642
(Iteration 3901 / 4900) loss: 1.465247
(Epoch 8 / 10) train acc: 0.549000; val_acc: 0.508000
(Iteration 4001 / 4900) loss: 1.354352
(Iteration 4101 / 4900) loss: 1.250002
(Iteration 4201 / 4900) loss: 1.307254
(Iteration 4301 / 4900) loss: 1.429705
(Iteration 4401 / 4900) loss: 1.496019
(Epoch 9 / 10) train acc: 0.547000; val_acc: 0.502000
(Iteration 4501 / 4900) loss: 1.302301
(Iteration 4601 / 4900) loss: 1.242352
(Iteration 4701 / 4900) loss: 1.336938
(Iteration 4801 / 4900) loss: 1.223351
(Epoch 10 / 10) train acc: 0.541000; val_acc: 0.517000
(Iteration 1 / 4900) loss: 2.302493
(Epoch 0 / 10) train acc: 0.088000; val_acc: 0.102000
(Iteration 101 / 4900) loss: 2.261172
(Iteration 201 / 4900) loss: 2.139410
```

```
(Iteration 301 / 4900) loss: 2.119160
(Iteration 401 / 4900) loss: 1.966803
(Epoch 1 / 10) train acc: 0.312000; val_acc: 0.301000
(Iteration 501 / 4900) loss: 1.870736
(Iteration 601 / 4900) loss: 1.869923
(Iteration 701 / 4900) loss: 1.881803
(Iteration 801 / 4900) loss: 1.792434
(Iteration 901 / 4900) loss: 1.736137
(Epoch 2 / 10) train acc: 0.333000; val_acc: 0.348000
(Iteration 1001 / 4900) loss: 1.795963
(Iteration 1101 / 4900) loss: 1.849129
(Iteration 1201 / 4900) loss: 1.871154
(Iteration 1301 / 4900) loss: 1.756509
(Iteration 1401 / 4900) loss: 1.724358
(Epoch 3 / 10) train acc: 0.362000; val_acc: 0.373000
(Iteration 1501 / 4900) loss: 1.761446
(Iteration 1601 / 4900) loss: 1.798741
(Iteration 1701 / 4900) loss: 1.740735
(Iteration 1801 / 4900) loss: 1.800022
(Iteration 1901 / 4900) loss: 1.730886
(Epoch 4 / 10) train acc: 0.400000; val_acc: 0.403000
(Iteration 2001 / 4900) loss: 1.719164
(Iteration 2101 / 4900) loss: 1.896652
(Iteration 2201 / 4900) loss: 1.819416
(Iteration 2301 / 4900) loss: 1.920386
(Iteration 2401 / 4900) loss: 1.723961
(Epoch 5 / 10) train acc: 0.396000; val_acc: 0.415000
(Iteration 2501 / 4900) loss: 1.731557
(Iteration 2601 / 4900) loss: 1.721073
(Iteration 2701 / 4900) loss: 1.677904
(Iteration 2801 / 4900) loss: 1.650564
(Iteration 2901 / 4900) loss: 1.764968
(Epoch 6 / 10) train acc: 0.416000; val_acc: 0.418000
(Iteration 3001 / 4900) loss: 1.779867
(Iteration 3101 / 4900) loss: 1.735192
(Iteration 3201 / 4900) loss: 1.595227
(Iteration 3301 / 4900) loss: 1.714974
(Iteration 3401 / 4900) loss: 1.713126
(Epoch 7 / 10) train acc: 0.407000; val_acc: 0.428000
(Iteration 3501 / 4900) loss: 1.595114
(Iteration 3601 / 4900) loss: 1.594025
(Iteration 3701 / 4900) loss: 1.563970
(Iteration 3801 / 4900) loss: 1.568630
(Iteration 3901 / 4900) loss: 1.636656
(Epoch 8 / 10) train acc: 0.420000; val_acc: 0.430000
(Iteration 4001 / 4900) loss: 1.606894
(Iteration 4101 / 4900) loss: 1.737844
(Iteration 4201 / 4900) loss: 1.488315
```

```
(Iteration 4301 / 4900) loss: 1.746451
(Iteration 4401 / 4900) loss: 1.577573
(Epoch 9 / 10) train acc: 0.419000; val_acc: 0.433000
(Iteration 4501 / 4900) loss: 1.629253
(Iteration 4601 / 4900) loss: 1.531396
(Iteration 4701 / 4900) loss: 1.686246
(Iteration 4801 / 4900) loss: 1.649778
(Epoch 10 / 10) train acc: 0.414000; val_acc: 0.432000
(Iteration 1 / 4900) loss: 2.301894
(Epoch 0 / 10) train acc: 0.080000; val_acc: 0.083000
(Iteration 101 / 4900) loss: 2.262181
(Iteration 201 / 4900) loss: 2.120140
(Iteration 301 / 4900) loss: 2.072150
(Iteration 401 / 4900) loss: 1.959002
(Epoch 1 / 10) train acc: 0.286000; val_acc: 0.309000
(Iteration 501 / 4900) loss: 1.950422
(Iteration 601 / 4900) loss: 1.957707
(Iteration 701 / 4900) loss: 1.853944
(Iteration 801 / 4900) loss: 1.831906
(Iteration 901 / 4900) loss: 1.762644
(Epoch 2 / 10) train acc: 0.345000; val_acc: 0.361000
(Iteration 1001 / 4900) loss: 1.827287
(Iteration 1101 / 4900) loss: 1.777763
(Iteration 1201 / 4900) loss: 1.866875
(Iteration 1301 / 4900) loss: 1.709601
(Iteration 1401 / 4900) loss: 2.008434
(Epoch 3 / 10) train acc: 0.365000; val_acc: 0.374000
(Iteration 1501 / 4900) loss: 1.662393
(Iteration 1601 / 4900) loss: 1.718024
(Iteration 1701 / 4900) loss: 1.801714
(Iteration 1801 / 4900) loss: 1.847742
(Iteration 1901 / 4900) loss: 1.709649
(Epoch 4 / 10) train acc: 0.373000; val_acc: 0.387000
(Iteration 2001 / 4900) loss: 1.771987
(Iteration 2101 / 4900) loss: 1.801081
(Iteration 2201 / 4900) loss: 1.633752
(Iteration 2301 / 4900) loss: 1.711961
(Iteration 2401 / 4900) loss: 1.582553
(Epoch 5 / 10) train acc: 0.409000; val_acc: 0.405000
(Iteration 2501 / 4900) loss: 1.701114
(Iteration 2601 / 4900) loss: 1.683802
(Iteration 2701 / 4900) loss: 1.746666
(Iteration 2801 / 4900) loss: 1.716508
(Iteration 2901 / 4900) loss: 1.665585
(Epoch 6 / 10) train acc: 0.424000; val_acc: 0.410000
(Iteration 3001 / 4900) loss: 1.820097
(Iteration 3101 / 4900) loss: 1.724546
(Iteration 3201 / 4900) loss: 1.714591
```

```
(Iteration 3301 / 4900) loss: 1.665061
(Iteration 3401 / 4900) loss: 1.562293
(Epoch 7 / 10) train acc: 0.459000; val_acc: 0.421000
(Iteration 3501 / 4900) loss: 1.548649
(Iteration 3601 / 4900) loss: 1.611329
(Iteration 3701 / 4900) loss: 1.669332
(Iteration 3801 / 4900) loss: 1.601656
(Iteration 3901 / 4900) loss: 1.641175
(Epoch 8 / 10) train acc: 0.429000; val_acc: 0.421000
(Iteration 4001 / 4900) loss: 1.496856
(Iteration 4101 / 4900) loss: 1.694367
(Iteration 4201 / 4900) loss: 1.669253
(Iteration 4301 / 4900) loss: 1.664244
(Iteration 4401 / 4900) loss: 1.614393
(Epoch 9 / 10) train acc: 0.436000; val_acc: 0.426000
(Iteration 4501 / 4900) loss: 1.630676
(Iteration 4601 / 4900) loss: 1.522713
(Iteration 4701 / 4900) loss: 1.583597
(Iteration 4801 / 4900) loss: 1.573687
(Epoch 10 / 10) train acc: 0.421000; val_acc: 0.432000
(Iteration 1 / 4900) loss: 2.303755
(Epoch 0 / 10) train acc: 0.121000; val_acc: 0.137000
(Iteration 101 / 4900) loss: 2.245839
(Iteration 201 / 4900) loss: 2.196449
(Iteration 301 / 4900) loss: 2.019986
(Iteration 401 / 4900) loss: 1.998952
(Epoch 1 / 10) train acc: 0.293000; val_acc: 0.297000
(Iteration 501 / 4900) loss: 1.995212
(Iteration 601 / 4900) loss: 1.981287
(Iteration 701 / 4900) loss: 1.926303
(Iteration 801 / 4900) loss: 1.813835
(Iteration 901 / 4900) loss: 1.775772
(Epoch 2 / 10) train acc: 0.343000; val_acc: 0.353000
(Iteration 1001 / 4900) loss: 1.942830
(Iteration 1101 / 4900) loss: 1.971827
(Iteration 1201 / 4900) loss: 1.610022
(Iteration 1301 / 4900) loss: 1.746039
(Iteration 1401 / 4900) loss: 1.700404
(Epoch 3 / 10) train acc: 0.384000; val_acc: 0.378000
(Iteration 1501 / 4900) loss: 1.635120
(Iteration 1601 / 4900) loss: 1.800309
(Iteration 1701 / 4900) loss: 1.733592
(Iteration 1801 / 4900) loss: 1.710801
(Iteration 1901 / 4900) loss: 1.786710
(Epoch 4 / 10) train acc: 0.382000; val_acc: 0.388000
(Iteration 2001 / 4900) loss: 1.718729
(Iteration 2101 / 4900) loss: 1.751177
(Iteration 2201 / 4900) loss: 1.741436
```

```
(Iteration 2301 / 4900) loss: 1.576027
(Iteration 2401 / 4900) loss: 1.647483
(Epoch 5 / 10) train acc: 0.390000; val_acc: 0.408000
(Iteration 2501 / 4900) loss: 1.557211
(Iteration 2601 / 4900) loss: 1.765059
(Iteration 2701 / 4900) loss: 1.863196
(Iteration 2801 / 4900) loss: 1.589997
(Iteration 2901 / 4900) loss: 1.598828
(Epoch 6 / 10) train acc: 0.403000; val_acc: 0.412000
(Iteration 3001 / 4900) loss: 1.763887
(Iteration 3101 / 4900) loss: 1.559468
(Iteration 3201 / 4900) loss: 1.717065
(Iteration 3301 / 4900) loss: 1.828987
(Iteration 3401 / 4900) loss: 1.721685
(Epoch 7 / 10) train acc: 0.414000; val_acc: 0.413000
(Iteration 3501 / 4900) loss: 1.760467
(Iteration 3601 / 4900) loss: 1.537740
(Iteration 3701 / 4900) loss: 1.708315
(Iteration 3801 / 4900) loss: 1.525204
(Iteration 3901 / 4900) loss: 1.521524
(Epoch 8 / 10) train acc: 0.416000; val_acc: 0.418000
(Iteration 4001 / 4900) loss: 1.652965
(Iteration 4101 / 4900) loss: 1.713257
(Iteration 4201 / 4900) loss: 1.656675
(Iteration 4301 / 4900) loss: 1.612428
(Iteration 4401 / 4900) loss: 1.586736
(Epoch 9 / 10) train acc: 0.441000; val_acc: 0.426000
(Iteration 4501 / 4900) loss: 1.634101
(Iteration 4601 / 4900) loss: 1.588091
(Iteration 4701 / 4900) loss: 1.685373
(Iteration 4801 / 4900) loss: 1.554511
(Epoch 10 / 10) train acc: 0.445000; val_acc: 0.423000
test accuracy:  0.426
```

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.

[27]:
```
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:  0.519
```

[28]:
```
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:  0.495
```

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.

YourAnswer : 1, 3

YourExplanation : Testing accuracy much lower than training accuarcy suggests there could be overfitting in our model for the training data. Therefore, 1. This is a valid approach to take since our original dataset could be too small, and not be able to generalize; 2. This may not be helpful since more hidden layers would lead to more overfitting, as the neurons learn from the training data even better. 3. This is valid because regularization prevents the model to be too overfitting.

features

February 1, 2024

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

# TODO: Enter the foldername in your Drive where you have saved the unzipped
# assignment folder, e.g. 'cse493g1/assignments/assignment2/'
FOLDERNAME = 'cse493g1/assignments/assignment2/'
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 /content/drive/My\ Drive/$FOLDERNAME/cse493g1/datasets/
!bash get_datasets.sh
%cd /content/drive/My\ Drive/$FOLDERNAME
```

```
Mounted at /content/drive
/content/drive/My Drive/cse493g1/assignments/assignment2/cse493g1/datasets
/content/drive/My Drive/cse493g1/assignments/assignment2
```

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.

[7]:
```
import random
import numpy as np
from cse493g1.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.

[ ]:
```
from cse493g1.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 = 'cse493g1/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.

[9]:
```
from cse493g1.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))])
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-9-a73815e32ec9> in <cell line: 5>()
      3 num_color_bins = 10 # Number of bins in the color histogram
      4 feature_fns = [hog_feature, lambda img: color_histogram_hsv(img,␣
 ↪nbin=num_color_bins)]
----> 5 X_train_feats = extract_features(X_train, feature_fns, verbose=True)
      6 X_val_feats = extract_features(X_val, feature_fns)
      7 X_test_feats = extract_features(X_test, feature_fns)

NameError: name 'X_train' is not defined
```

1.3 Train SVM on features

Using the multiclass SVM code developed in Assignment 1 (copy and paste your code from Assignment 1 into the file `cse493g1/classifiers/linear_svm.py`), train SVMs on top of the features extracted above. This should achieve better results than training SVMs directly on top of raw pixels.

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

from cse493g1.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 0.43 on the validation set.       #
################################################################################
# *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
print(np.shape(X_train_feats))
for lr in learning_rates:
    for reg in regularization_strengths:
        svm = LinearSVM()
        svm.train(X_train_feats, y_train, learning_rate=lr, reg=reg,␣
 ↪num_iters=1000)
        y_train_pred = svm.predict(X_train_feats)
```

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

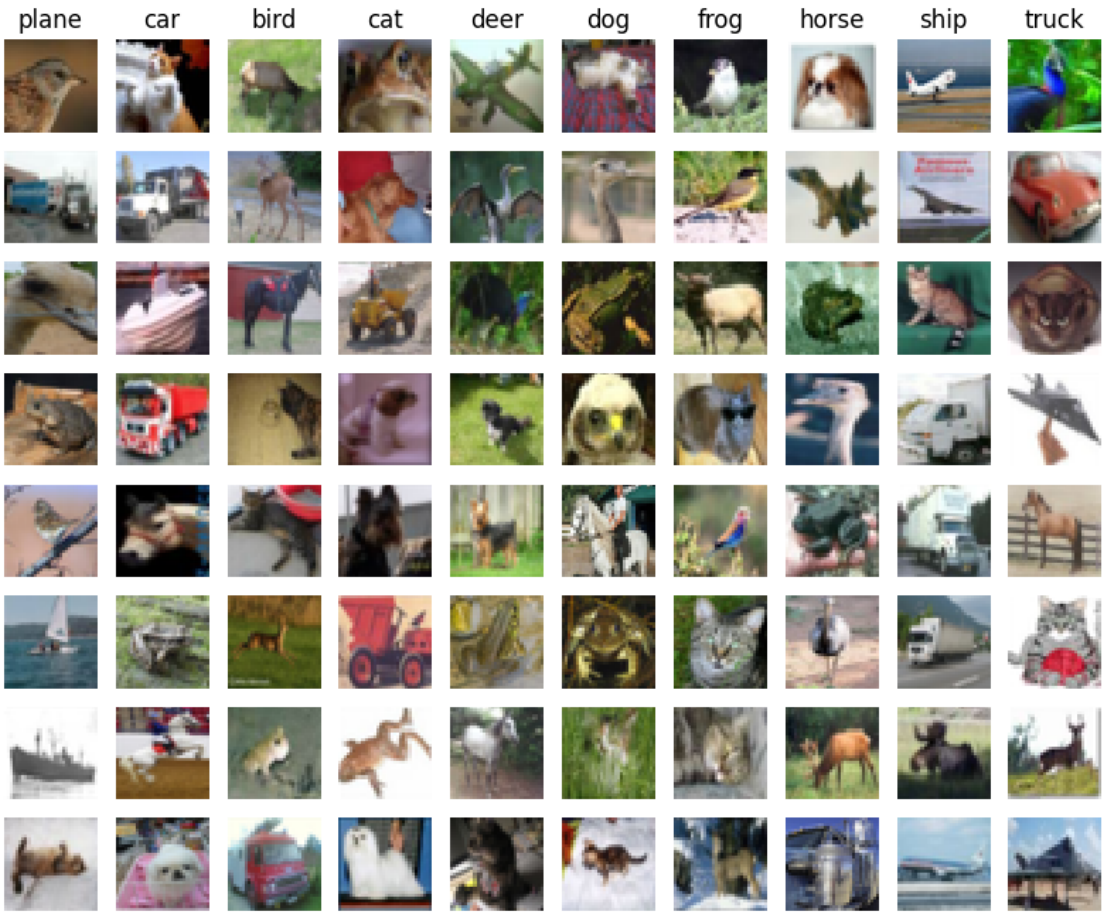
```
(49000, 155)
lr 1.000000e-09 reg 5.000000e+04 train accuracy: 0.101551 val accuracy: 0.092000
lr 1.000000e-09 reg 5.000000e+05 train accuracy: 0.091735 val accuracy: 0.090000
lr 1.000000e-09 reg 5.000000e+06 train accuracy: 0.290551 val accuracy: 0.303000
lr 1.000000e-08 reg 5.000000e+04 train accuracy: 0.091816 val accuracy: 0.093000
lr 1.000000e-08 reg 5.000000e+05 train accuracy: 0.413714 val accuracy: 0.402000
lr 1.000000e-08 reg 5.000000e+06 train accuracy: 0.397327 val accuracy: 0.392000
lr 1.000000e-07 reg 5.000000e+04 train accuracy: 0.415041 val accuracy: 0.418000
lr 1.000000e-07 reg 5.000000e+05 train accuracy: 0.409980 val accuracy: 0.411000
lr 1.000000e-07 reg 5.000000e+06 train accuracy: 0.289347 val accuracy: 0.285000
best validation accuracy achieved: 0.418000
```

[ ]:
```
# 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)
```

```
0.424
```

[ ]:
```
# 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?

YourAnswer : The misclassified results make sense. We can find features that the misclassified class may have similar to the correct class. For example, the ships and trucks look similar to planes, so it make some sense if the network mis-labeled them as planes. Also, in training sets,

bird pictures tend to have a blue sky as background, so the model tends to put some other stuff with blue backgrounds under the bird class.

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.

[ ]:
```
# 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)
```

[ ]:
```
from cse493g1.classifiers.fc_net import TwoLayerNet
from cse493g1.solver import Solver

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

data = {
    'X_train': X_train_feats,
    'y_train': y_train,
    'X_val': X_val_feats,
    'y_val': y_val,
    'X_test': X_test_feats,
    'y_test': y_test,
}

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)*****

learning_rates = [5e-2]
learning_rate_decays = [0.95, 0.9, 0.8]
best_acc = -1

for lr in learning_rates:
    for decay in learning_rate_decays:
        # Create a new Solver instance for the current model and dataset
        solver = Solver(net, data,
                        update_rule='sgd',
                        optim_config={
                            'learning_rate': lr,
                        },
                        lr_decay=decay,
                        num_epochs=10, batch_size=100,
                        print_every=200)
        # Train the model
        solver.train()
        if solver.best_val_acc>best_acc:
          best_acc = solver.best_val_acc
          best_net = net

print('test accuracy: ', solver.best_val_acc)
# *****END OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
```

```
(Iteration 1 / 4900) loss: 2.302566
(Epoch 0 / 10) train acc: 0.099000; val_acc: 0.078000
(Iteration 201 / 4900) loss: 2.221047
(Iteration 401 / 4900) loss: 1.699845
(Epoch 1 / 10) train acc: 0.404000; val_acc: 0.385000
(Iteration 601 / 4900) loss: 1.460967
(Iteration 801 / 4900) loss: 1.458902
(Epoch 2 / 10) train acc: 0.521000; val_acc: 0.507000
(Iteration 1001 / 4900) loss: 1.503822
(Iteration 1201 / 4900) loss: 1.308927
(Iteration 1401 / 4900) loss: 1.456935
(Epoch 3 / 10) train acc: 0.493000; val_acc: 0.506000
(Iteration 1601 / 4900) loss: 1.386910
(Iteration 1801 / 4900) loss: 1.382462
(Epoch 4 / 10) train acc: 0.516000; val_acc: 0.517000
(Iteration 2001 / 4900) loss: 1.372107
(Iteration 2201 / 4900) loss: 1.186276
(Iteration 2401 / 4900) loss: 1.155913
(Epoch 5 / 10) train acc: 0.564000; val_acc: 0.527000
(Iteration 2601 / 4900) loss: 1.370594
```

```
(Iteration 2801 / 4900) loss: 1.271748
(Epoch 6 / 10) train acc: 0.577000; val_acc: 0.536000
(Iteration 3001 / 4900) loss: 1.418419
(Iteration 3201 / 4900) loss: 1.367008
(Iteration 3401 / 4900) loss: 1.382903
(Epoch 7 / 10) train acc: 0.573000; val_acc: 0.538000
(Iteration 3601 / 4900) loss: 1.215988
(Iteration 3801 / 4900) loss: 1.246382
(Epoch 8 / 10) train acc: 0.552000; val_acc: 0.541000
(Iteration 4001 / 4900) loss: 1.333049
(Iteration 4201 / 4900) loss: 1.305666
(Iteration 4401 / 4900) loss: 1.048863
(Epoch 9 / 10) train acc: 0.586000; val_acc: 0.561000
(Iteration 4601 / 4900) loss: 1.074458
(Iteration 4801 / 4900) loss: 1.161997
(Epoch 10 / 10) train acc: 0.589000; val_acc: 0.553000
(Iteration 1 / 4900) loss: 1.073182
(Epoch 0 / 10) train acc: 0.578000; val_acc: 0.556000
(Iteration 201 / 4900) loss: 0.935288
(Iteration 401 / 4900) loss: 1.151092
(Epoch 1 / 10) train acc: 0.620000; val_acc: 0.555000
(Iteration 601 / 4900) loss: 1.015116
(Iteration 801 / 4900) loss: 1.099137
(Epoch 2 / 10) train acc: 0.616000; val_acc: 0.566000
(Iteration 1001 / 4900) loss: 1.115380
(Iteration 1201 / 4900) loss: 1.114873
(Iteration 1401 / 4900) loss: 0.983820
(Epoch 3 / 10) train acc: 0.607000; val_acc: 0.576000
(Iteration 1601 / 4900) loss: 1.181057
(Iteration 1801 / 4900) loss: 0.937757
(Epoch 4 / 10) train acc: 0.638000; val_acc: 0.578000
(Iteration 2001 / 4900) loss: 0.924119
(Iteration 2201 / 4900) loss: 1.124247
(Iteration 2401 / 4900) loss: 0.924593
(Epoch 5 / 10) train acc: 0.634000; val_acc: 0.579000
(Iteration 2601 / 4900) loss: 0.958110
(Iteration 2801 / 4900) loss: 0.817273
(Epoch 6 / 10) train acc: 0.661000; val_acc: 0.586000
(Iteration 3001 / 4900) loss: 0.993645
(Iteration 3201 / 4900) loss: 1.089167
(Iteration 3401 / 4900) loss: 1.080946
(Epoch 7 / 10) train acc: 0.638000; val_acc: 0.591000
(Iteration 3601 / 4900) loss: 0.926380
(Iteration 3801 / 4900) loss: 1.037851
(Epoch 8 / 10) train acc: 0.673000; val_acc: 0.589000
(Iteration 4001 / 4900) loss: 0.854877
(Iteration 4201 / 4900) loss: 1.029091
(Iteration 4401 / 4900) loss: 0.801275
```

```
(Epoch 9 / 10) train acc: 0.662000; val_acc: 0.597000
(Iteration 4601 / 4900) loss: 0.980301
(Iteration 4801 / 4900) loss: 0.948075
(Epoch 10 / 10) train acc: 0.674000; val_acc: 0.583000
(Iteration 1 / 4900) loss: 1.003705
(Epoch 0 / 10) train acc: 0.652000; val_acc: 0.594000
(Iteration 201 / 4900) loss: 0.922478
(Iteration 401 / 4900) loss: 0.804009
(Epoch 1 / 10) train acc: 0.663000; val_acc: 0.588000
(Iteration 601 / 4900) loss: 0.828927
(Iteration 801 / 4900) loss: 0.881079
(Epoch 2 / 10) train acc: 0.654000; val_acc: 0.593000
(Iteration 1001 / 4900) loss: 0.900916
(Iteration 1201 / 4900) loss: 0.911862
(Iteration 1401 / 4900) loss: 0.990567
(Epoch 3 / 10) train acc: 0.714000; val_acc: 0.583000
(Iteration 1601 / 4900) loss: 1.053716
(Iteration 1801 / 4900) loss: 0.783302
(Epoch 4 / 10) train acc: 0.717000; val_acc: 0.591000
(Iteration 2001 / 4900) loss: 0.780955
(Iteration 2201 / 4900) loss: 0.862460
(Iteration 2401 / 4900) loss: 0.942033
(Epoch 5 / 10) train acc: 0.670000; val_acc: 0.594000
(Iteration 2601 / 4900) loss: 1.004265
(Iteration 2801 / 4900) loss: 0.941386
(Epoch 6 / 10) train acc: 0.697000; val_acc: 0.603000
(Iteration 3001 / 4900) loss: 0.807103
(Iteration 3201 / 4900) loss: 0.872875
(Iteration 3401 / 4900) loss: 1.085279
(Epoch 7 / 10) train acc: 0.695000; val_acc: 0.590000
(Iteration 3601 / 4900) loss: 0.835120
(Iteration 3801 / 4900) loss: 0.965538
(Epoch 8 / 10) train acc: 0.700000; val_acc: 0.606000
(Iteration 4001 / 4900) loss: 0.844577
(Iteration 4201 / 4900) loss: 1.093245
(Iteration 4401 / 4900) loss: 1.017494
(Epoch 9 / 10) train acc: 0.695000; val_acc: 0.600000
(Iteration 4601 / 4900) loss: 0.790880
(Iteration 4801 / 4900) loss: 0.962180
(Epoch 10 / 10) train acc: 0.715000; val_acc: 0.596000
test accuracy:  0.606
```

[5]:
```
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()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-5-f035189be89d> in <cell line: 1>()
----> 1 plt.subplot(2, 1, 1)
      2 plt.title('Training loss')
      3 plt.plot(solver.loss_history, 'o')
      4 plt.xlabel('Iteration')
      5

NameError: name 'plt' is not defined
```

[3]:
```
# 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)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-3-d1f5c9ab9c85> in <cell line: 4>()
      2 # to get more than 55% accuracy.
      3
----> 4 y_test_pred = np.argmax(best_net.loss(data['X_test']), axis=1)
      5 test_acc = (y_test_pred == data['y_test']).mean()
      6 print(test_acc)

NameError: name 'np' is not defined
```

FullyConnectedNets

February 1, 2024

[71]:
```
# This mounts your Google Drive to the Colab VM.
from google.colab import drive
drive.mount('/content/drive')

# TODO: Enter the foldername in your Drive where you have saved the unzipped
# assignment folder, e.g. 'cse493g1/assignments/assignment2/'
FOLDERNAME = 'cse493g1/assignments/assignment2/'
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 /content/drive/My\ Drive/$FOLDERNAME/cse493g1/datasets/
!bash get_datasets.sh
%cd /content/drive/My\ Drive/$FOLDERNAME
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
/content/drive/My Drive/cse493g1/assignments/assignment2/cse493g1/datasets
/content/drive/My Drive/cse493g1/assignments/assignment2
```

1 Multi-Layer Fully Connected Network

In this exercise, you will implement a fully connected network with an arbitrary number of hidden layers.

Read through the `FullyConnectedNet` class in the file `cse493g1/classifiers/fc_net.py`.

Implement the network initialization, forward pass, and backward pass. Throughout this assignment, you will be implementing layers in `cse493g1/layers.py`. You can re-use your implementations for `affine_forward`, `affine_backward`, `relu_forward`, `relu_backward`, and `softmax_loss` from the previous notebook.

[72]:
```
# Setup cell.
import time
import numpy as np
import matplotlib.pyplot as plt
from cse493g1.classifiers.fc_net import *
from cse493g1.data_utils import get_CIFAR10_data
from cse493g1.gradient_check import eval_numerical_gradient,␣
 ↪eval_numerical_gradient_array
from cse493g1.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"

%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))))
```

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

[73]:
```
# Load the (preprocessed) CIFAR-10 data.
data = get_CIFAR10_data()
for k, v in list(data.items()):
    print(f"{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,)
```

1.1 Initial Loss and Gradient Check

As a sanity check, run the following to check the initial loss and to gradient check the network both with and without regularization. This is a good way to see if the initial losses seem reasonable.

For gradient checking, you should expect to see errors around 1e-7 or less.

[74]:
```
np.random.seed(493)
N, D, H1, H2, C = 2, 15, 20, 30, 10
X = np.random.randn(N, D)
y = np.random.randint(C, size=(N,))
```

```
for reg in [0, 3.14]:
    print("Running check with reg = ", reg)
    model = FullyConnectedNet(
        [H1, H2],
        input_dim=D,
        num_classes=C,
        reg=reg,
        weight_scale=5e-2,
        dtype=np.float64
    )

    loss, grads = model.loss(X, y)
    print("Initial loss: ", loss)

    # Most of the errors should be on the order of e-7 or smaller.
    # NOTE: It is fine however to see an error for W2 on the order of e-5
    # for the check when reg = 0.0
    for name in sorted(grads):
        f = lambda _: model.loss(X, y)[0]
        grad_num = eval_numerical_gradient(f, model.params[name],␣
 ↪verbose=False, h=1e-5)
        print(f"{name} relative error: {rel_error(grad_num, grads[name])}")
```

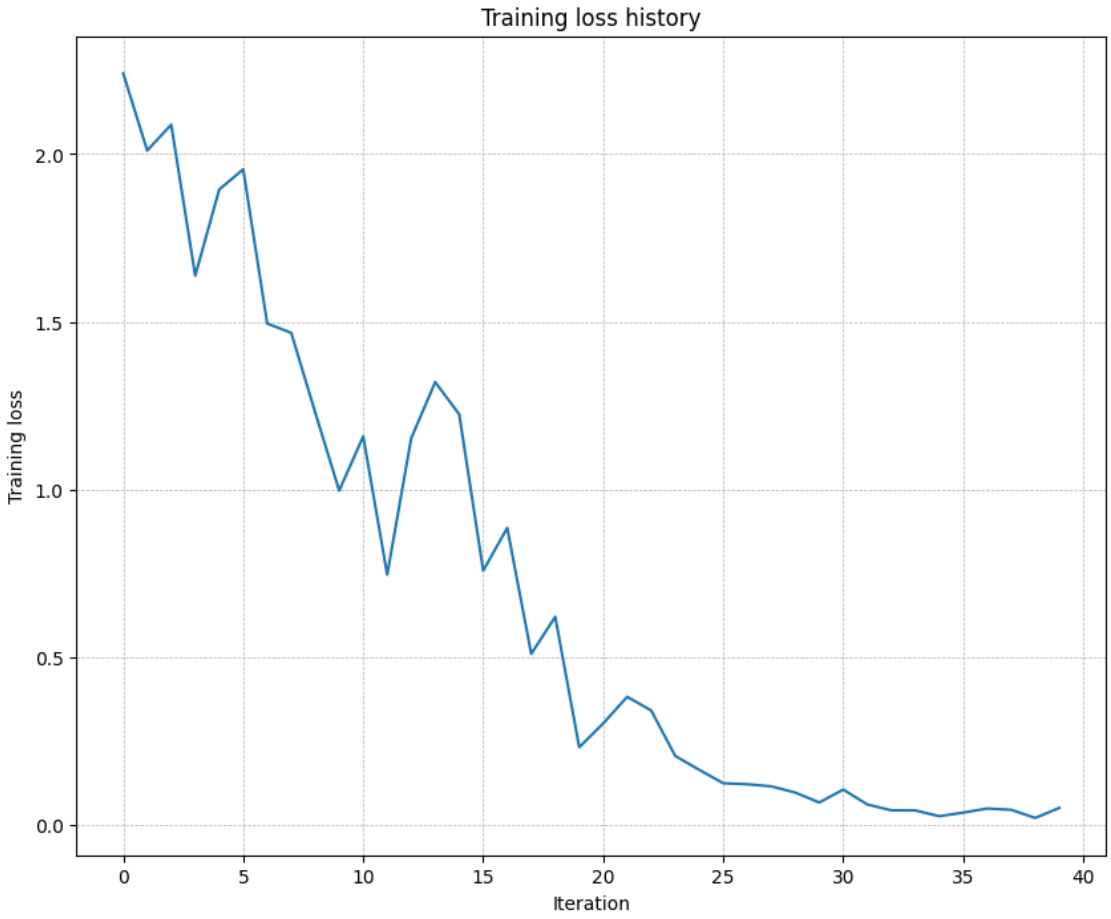
```
Running check with reg =  0
Initial loss:  2.299821914918452
W1 relative error: 9.656953386228676e-08
W2 relative error: 4.42914245511279e-06
W3 relative error: 1.4112116043233687e-06
b1 relative error: 9.10105007110364e-09
b2 relative error: 1.3597116499374134e-07
b3 relative error: 1.3432054300218008e-10
Running check with reg =  3.14
Initial loss:  6.991659719884911
W1 relative error: 2.2442521861888318e-08
W2 relative error: 1.3072726683559537e-08
W3 relative error: 1.585488123924185e-08
b1 relative error: 4.368076841105884e-08
b2 relative error: 1.0110675182132624e-08
b3 relative error: 1.1311009016240723e-10
```

As another sanity check, make sure your network can overfit on a small dataset of 50 images. First, we will try a three-layer network with 100 units in each hidden layer. In the following cell, tweak the **learning rate** and **weight initialization scale** to overfit and achieve 100% training accuracy within 20 epochs.

[75]:
```
# TODO: Use a three-layer Net to overfit 50 training examples by
# tweaking just the learning rate and initialization scale.

num_train = 50
small_data = {
  "X_train": data["X_train"][:num_train],
  "y_train": data["y_train"][:num_train],
  "X_val": data["X_val"],
  "y_val": data["y_val"],
}

weight_scale = 1e-2   # Experiment with this!
learning_rate = 1e-2  # Experiment with this!
model = FullyConnectedNet(
    [100, 100],
    weight_scale=weight_scale,
    dtype=np.float64
)
solver = Solver(
    model,
    small_data,
    print_every=10,
    num_epochs=20,
    batch_size=25,
    update_rule="sgd",
    optim_config={"learning_rate": learning_rate},
)
solver.train()

plt.plot(solver.loss_history)
plt.title("Training loss history")
plt.xlabel("Iteration")
plt.ylabel("Training loss")
plt.grid(linestyle='--', linewidth=0.5)
plt.show()
```

```
(Iteration 1 / 40) loss: 2.240372
(Epoch 0 / 20) train acc: 0.320000; val_acc: 0.117000
(Epoch 1 / 20) train acc: 0.240000; val_acc: 0.106000
(Epoch 2 / 20) train acc: 0.360000; val_acc: 0.132000
(Epoch 3 / 20) train acc: 0.520000; val_acc: 0.120000
(Epoch 4 / 20) train acc: 0.560000; val_acc: 0.157000
(Epoch 5 / 20) train acc: 0.640000; val_acc: 0.172000
(Iteration 11 / 40) loss: 1.158698
(Epoch 6 / 20) train acc: 0.680000; val_acc: 0.173000
(Epoch 7 / 20) train acc: 0.740000; val_acc: 0.191000
(Epoch 8 / 20) train acc: 0.820000; val_acc: 0.161000
```

```
(Epoch 9 / 20) train acc: 0.960000; val_acc: 0.193000
(Epoch 10 / 20) train acc: 0.960000; val_acc: 0.200000
(Iteration 21 / 40) loss: 0.303750
(Epoch 11 / 20) train acc: 0.980000; val_acc: 0.176000
(Epoch 12 / 20) train acc: 1.000000; val_acc: 0.186000
(Epoch 13 / 20) train acc: 0.980000; val_acc: 0.191000
(Epoch 14 / 20) train acc: 1.000000; val_acc: 0.188000
(Epoch 15 / 20) train acc: 1.000000; val_acc: 0.216000
(Iteration 31 / 40) loss: 0.106444
(Epoch 16 / 20) train acc: 1.000000; val_acc: 0.194000
(Epoch 17 / 20) train acc: 1.000000; val_acc: 0.194000
(Epoch 18 / 20) train acc: 1.000000; val_acc: 0.190000
(Epoch 19 / 20) train acc: 1.000000; val_acc: 0.185000
(Epoch 20 / 20) train acc: 0.960000; val_acc: 0.203000
```

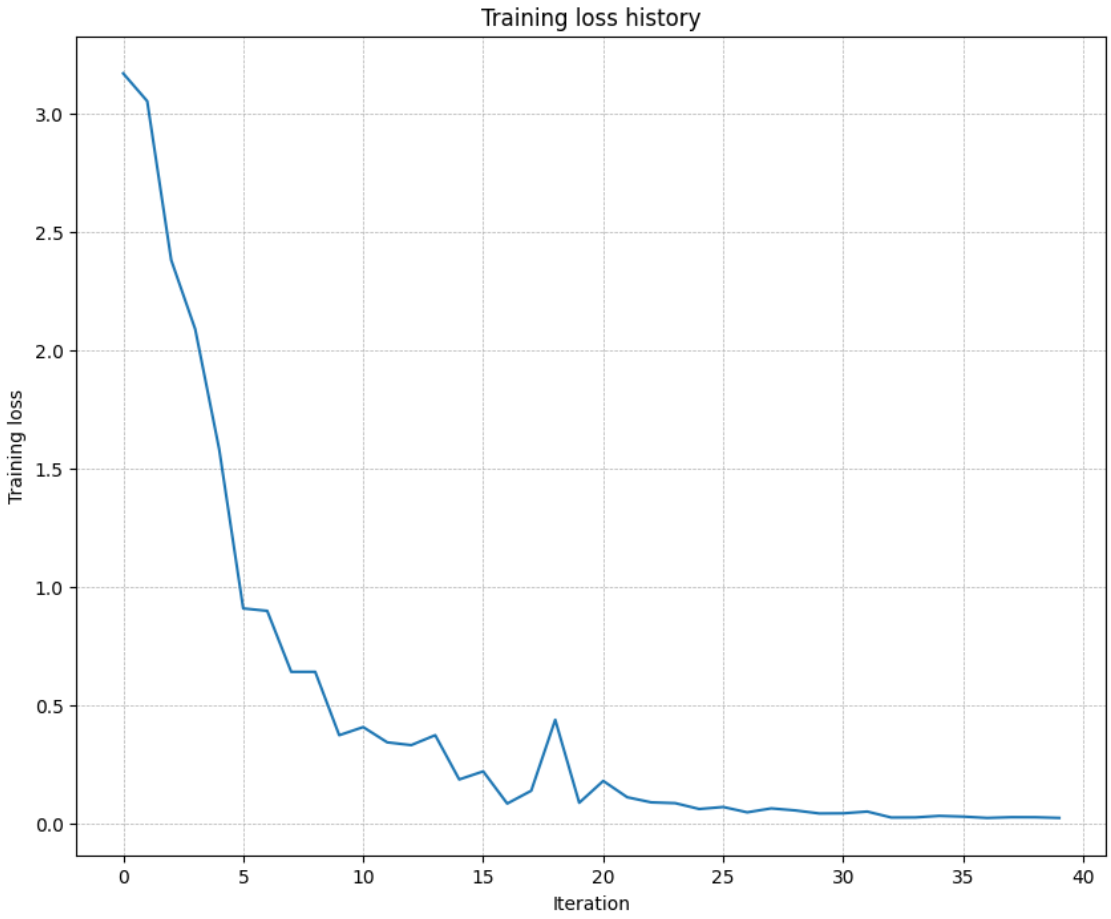


Now, try to use a five-layer network with 100 units on each layer to overfit on 50 training examples. Again, you will have to adjust the learning rate and weight initialization scale, but you should be able to achieve 100% training accuracy within 20 epochs.

[76]:
```
# TODO: Use a five-layer Net to overfit 50 training examples by
# tweaking just the learning rate and initialization scale.

num_train = 50
small_data = {
  'X_train': data['X_train'][:num_train],
  'y_train': data['y_train'][:num_train],
  'X_val': data['X_val'],
  'y_val': data['y_val'],
}

learning_rate = 1e-2  # Experiment with this!
weight_scale = 5e-2   # Experiment with this!
model = FullyConnectedNet(
    [100, 100, 100, 100],
    weight_scale=weight_scale,
    dtype=np.float64
)
solver = Solver(
    model,
    small_data,
    print_every=10,
    num_epochs=20,
    batch_size=25,
    update_rule='sgd',
    optim_config={'learning_rate': learning_rate},
)
solver.train()

plt.plot(solver.loss_history)
plt.title('Training loss history')
plt.xlabel('Iteration')
plt.ylabel('Training loss')
plt.grid(linestyle='--', linewidth=0.5)
plt.show()
```

```
(Iteration 1 / 40) loss: 3.168202
(Epoch 0 / 20) train acc: 0.280000; val_acc: 0.123000
(Epoch 1 / 20) train acc: 0.240000; val_acc: 0.120000
(Epoch 2 / 20) train acc: 0.620000; val_acc: 0.113000
(Epoch 3 / 20) train acc: 0.820000; val_acc: 0.132000
(Epoch 4 / 20) train acc: 0.840000; val_acc: 0.135000
(Epoch 5 / 20) train acc: 0.880000; val_acc: 0.146000
(Iteration 11 / 40) loss: 0.408332
(Epoch 6 / 20) train acc: 0.960000; val_acc: 0.160000
(Epoch 7 / 20) train acc: 0.960000; val_acc: 0.157000
(Epoch 8 / 20) train acc: 0.980000; val_acc: 0.157000
```

```
(Epoch 9 / 20) train acc: 0.980000; val_acc: 0.159000
(Epoch 10 / 20) train acc: 0.980000; val_acc: 0.157000
(Iteration 21 / 40) loss: 0.180979
(Epoch 11 / 20) train acc: 1.000000; val_acc: 0.144000
(Epoch 12 / 20) train acc: 1.000000; val_acc: 0.151000
(Epoch 13 / 20) train acc: 1.000000; val_acc: 0.150000
(Epoch 14 / 20) train acc: 1.000000; val_acc: 0.153000
(Epoch 15 / 20) train acc: 1.000000; val_acc: 0.152000
(Iteration 31 / 40) loss: 0.044178
(Epoch 16 / 20) train acc: 1.000000; val_acc: 0.156000
(Epoch 17 / 20) train acc: 1.000000; val_acc: 0.156000
(Epoch 18 / 20) train acc: 1.000000; val_acc: 0.160000
(Epoch 19 / 20) train acc: 1.000000; val_acc: 0.160000
(Epoch 20 / 20) train acc: 1.000000; val_acc: 0.161000
```



1.2 Inline Question 1:

Did you notice anything about the comparative difficulty of training the three-layer network vs. training the five-layer network? In particular, based on your experience, which network seemed

more sensitive to the initialization scale? Why do you think that is the case?

1.3 Answer:

I noticed that it is more difficult to train the five-layer network. The five-layer network is more sensitive to the initialization scale, becasue when tuning the parameter, I spend more time and need to pay more attention to the value I input for the five-layer network, otherwise the training accuracy would not change in a noticeable way as the iteration goes, while that's not the case for the three-layer network. That may be due to the gradients being too small through the backpropagation.

2 Update rules

So far we have used vanilla stochastic gradient descent (SGD) as our update rule. More sophisticated update rules can make it easier to train deep networks. We will implement a few of the most commonly used update rules and compare them to vanilla SGD.

2.1 SGD+Momentum

Stochastic gradient descent with momentum is a widely used update rule that tends to make deep networks converge faster than vanilla stochastic gradient descent. See the Momentum Update section for more information.

Open the file `cse493g1/optim.py` and read the documentation at the top of the file to make sure you understand the API. Implement the SGD+momentum update rule in the function `sgd_momentum` and run the following to check your implementation. You should see errors less than e-8.

[77]:
```
from cse493g1.optim import sgd_momentum

N, D = 4, 5
w = np.linspace(-0.4, 0.6, num=N*D).reshape(N, D)
dw = np.linspace(-0.6, 0.4, num=N*D).reshape(N, D)
v = np.linspace(0.6, 0.9, num=N*D).reshape(N, D)

config = {"learning_rate": 1e-3, "velocity": v}
next_w, _ = sgd_momentum(w, dw, config=config)

expected_next_w = np.asarray([
  [ 0.1406,      0.20738947,  0.27417895,  0.34096842,  0.40775789],
  [ 0.47454737,  0.54133684,  0.60812632,  0.67491579,  0.74170526],
  [ 0.80849474,  0.87528421,  0.94207368,  1.00886316,  1.07565263],
  [ 1.14244211,  1.20923158,  1.27602105,  1.34281053,  1.4096    ]])
expected_velocity = np.asarray([
  [ 0.5406,      0.55475789,  0.56891579, 0.58307368,  0.59723158],
  [ 0.61138947,  0.62554737,  0.63970526,  0.65386316,  0.66802105],
  [ 0.68217895,  0.69633684,  0.71049474,  0.72465263,  0.73881053],
  [ 0.75296842,  0.76712632,  0.78128421,  0.79544211,  0.8096    ]])

# Should see relative errors around e-8 or less
```

```
print("next_w error: ", rel_error(next_w, expected_next_w))
print("velocity error: ", rel_error(expected_velocity, config["velocity"]))
```

```
next_w error:  8.882347033505819e-09
velocity error:  4.269287743278663e-09
```

Once you have done so, run the following to train a six-layer network with both SGD and SGD+momentum. You should see the SGD+momentum update rule converge faster.

[98]:
```
num_train = 4000
small_data = {
  'X_train': data['X_train'][:num_train],
  'y_train': data['y_train'][:num_train],
  'X_val': data['X_val'],
  'y_val': data['y_val'],
}

solvers = {}

for update_rule in ['sgd', 'sgd_momentum']:
    print('Running with ', update_rule)
    model = FullyConnectedNet(
        [100, 100, 100, 100, 100],
        weight_scale=5e-2
    )

    solver = Solver(
        model,
        small_data,
        num_epochs=5,
        batch_size=100,
        update_rule=update_rule,
        optim_config={'learning_rate': 5e-3},
        verbose=True,
    )
    solvers[update_rule] = solver
    solver.train()

fig, axes = plt.subplots(3, 1, figsize=(15, 15))

axes[0].set_title('Training loss')
axes[0].set_xlabel('Iteration')
axes[1].set_title('Training accuracy')
axes[1].set_xlabel('Epoch')
axes[2].set_title('Validation accuracy')
axes[2].set_xlabel('Epoch')

for update_rule, solver in solvers.items():
```

```
    axes[0].plot(solver.loss_history, label=f"loss_{update_rule}")
    axes[1].plot(solver.train_acc_history, label=f"train_acc_{update_rule}")
    axes[2].plot(solver.val_acc_history, label=f"val_acc_{update_rule}")

for ax in axes:
    ax.legend(loc="best", ncol=4)
    ax.grid(linestyle='--', linewidth=0.5)

plt.show()
```

```
Running with  sgd
(Iteration 1 / 2450) loss: 2.602120
(Epoch 0 / 5) train acc: 0.105000; val_acc: 0.128000
(Iteration 11 / 2450) loss: 2.250900
(Iteration 21 / 2450) loss: 2.266915
(Iteration 31 / 2450) loss: 2.214251
(Iteration 41 / 2450) loss: 2.165780
(Iteration 51 / 2450) loss: 2.086675
(Iteration 61 / 2450) loss: 2.281985
(Iteration 71 / 2450) loss: 2.143804
(Iteration 81 / 2450) loss: 2.099932
(Iteration 91 / 2450) loss: 2.032412
(Iteration 101 / 2450) loss: 2.077370
(Iteration 111 / 2450) loss: 2.081330
(Iteration 121 / 2450) loss: 2.084799
(Iteration 131 / 2450) loss: 1.922381
(Iteration 141 / 2450) loss: 1.971535
(Iteration 151 / 2450) loss: 1.855558
(Iteration 161 / 2450) loss: 1.984556
(Iteration 171 / 2450) loss: 2.033749
(Iteration 181 / 2450) loss: 1.876934
(Iteration 191 / 2450) loss: 1.986743
(Iteration 201 / 2450) loss: 2.055625
(Iteration 211 / 2450) loss: 1.864582
(Iteration 221 / 2450) loss: 1.958278
(Iteration 231 / 2450) loss: 2.020546
(Iteration 241 / 2450) loss: 1.938985
(Iteration 251 / 2450) loss: 1.977451
(Iteration 261 / 2450) loss: 1.900168
(Iteration 271 / 2450) loss: 1.840328
(Iteration 281 / 2450) loss: 1.784010
(Iteration 291 / 2450) loss: 1.860527
(Iteration 301 / 2450) loss: 1.889478
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-98-5b14e5ee3eb6> in <cell line: 11>()
```

```
     26     )
     27     solvers[update_rule] = solver
---> 28     solver.train()
     29
     30 fig, axes = plt.subplots(3, 1, figsize=(15, 15))

/content/drive/My Drive/cse493g1/assignments/assignment2/cse493g1/solver.py in␣
 ↪train(self)
    261
    262         for t in range(num_iterations):
--> 263             self._step()
    264
    265             # Maybe print training loss

/content/drive/My Drive/cse493g1/assignments/assignment2/cse493g1/solver.py in␣
 ↪_step(self)
    179
    180         # Compute loss and gradient
--> 181         loss, grads = self.model.loss(X_batch, y_batch)
    182         self.loss_history.append(loss)
    183

/content/drive/My Drive/cse493g1/assignments/assignment2/cse493g1/classifiers/
 ↪fc_net.py in loss(self, X, y)
    286         # backward pass for all layers except the last one
    287         for i in range(self.num_layers - 1, 0, -1):  # iterate in␣
 ↪reverse order
--> 288             dout, grads["W" + str(i)], grads["b" + str(i)] =␣
 ↪affine_relu_backward(dout, caches[i - 1])
    289             grads["W" + str(i)] += self.reg * self.params["W" + str(i)]
    290

/content/drive/My Drive/cse493g1/assignments/assignment2/cse493g1/layer_utils.p␣
 ↪in affine_relu_backward(dout, cache)
     26     fc_cache, relu_cache = cache
     27     da = relu_backward(dout, relu_cache)
---> 28     dx, dw, db = affine_backward(da, fc_cache)
     29     return dx, dw, db
     30

/content/drive/My Drive/cse493g1/assignments/assignment2/cse493g1/layers.py in␣
 ↪affine_backward(dout, cache)
     70     dx = dx.reshape(x.shape)
     71     x_reshaped = x.reshape(N, -1) # NxD
---> 72     dw = x_reshaped.T.dot(dout) # DxN * NxM = DxM
     73     db = np.sum(dout, axis=0) # Mx1
     74
```

```
KeyboardInterrupt:
```

2.2 RMSProp and Adam

RMSProp [1] and Adam [2] are update rules that set per-parameter learning rates by using a running average of the second moments of gradients.

In the file `cse493g1/optim.py`, implement the RMSProp update rule in the `rmsprop` function and implement the Adam update rule in the `adam` function, and check your implementations using the tests below.

NOTE: Please implement the *complete* Adam update rule (with the bias correction mechanism), not the first simplified version mentioned in the course notes.

[1] Tijmen Tieleman and Geoffrey Hinton. "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude." COURSERA: Neural Networks for Machine Learning 4 (2012).

[2] Diederik Kingma and Jimmy Ba, "Adam: A Method for Stochastic Optimization", ICLR 2015.

[79]:
```
# Test RMSProp implementation
from cse493g1.optim import rmsprop

N, D = 4, 5
w = np.linspace(-0.4, 0.6, num=N*D).reshape(N, D)
dw = np.linspace(-0.6, 0.4, num=N*D).reshape(N, D)
cache = np.linspace(0.6, 0.9, num=N*D).reshape(N, D)

config = {'learning_rate': 1e-2, 'cache': cache}
next_w, _ = rmsprop(w, dw, config=config)

expected_next_w = np.asarray([
  [-0.39223849, -0.34037513, -0.28849239, -0.23659121, -0.18467247],
  [-0.132737,   -0.08078555, -0.02881884,  0.02316247,  0.07515774],
  [ 0.12716641,  0.17918792,  0.23122175,  0.28326742,  0.33532447],
  [ 0.38739248,  0.43947102,  0.49155973,  0.54365823,  0.59576619]])
expected_cache = np.asarray([
  [ 0.5976,      0.6126277,   0.6277108,   0.64284931,  0.65804321],
  [ 0.67329252,  0.68859723,  0.70395734,  0.71937285,  0.73484377],
  [ 0.75037008,  0.7659518,   0.78158892,  0.79728144,  0.81302936],
  [ 0.82883269,  0.84469141,  0.86060554,  0.87657507,  0.8926    ]])

# You should see relative errors around e-7 or less
print('next_w error: ', rel_error(expected_next_w, next_w))
print('cache error: ', rel_error(expected_cache, config['cache']))
```

```
next_w error:  9.524687511038133e-08
cache error:  2.6477955807156126e-09
```

[80]:
```
# Test Adam implementation
from cse493g1.optim import adam

N, D = 4, 5
w = np.linspace(-0.4, 0.6, num=N*D).reshape(N, D)
dw = np.linspace(-0.6, 0.4, num=N*D).reshape(N, D)
m = np.linspace(0.6, 0.9, num=N*D).reshape(N, D)
v = np.linspace(0.7, 0.5, num=N*D).reshape(N, D)

config = {'learning_rate': 1e-2, 'm': m, 'v': v, 't': 5}
next_w, _ = adam(w, dw, config=config)

expected_next_w = np.asarray([
  [-0.40094747, -0.34836187, -0.29577703, -0.24319299, -0.19060977],
  [-0.1380274,  -0.08544591, -0.03286534,  0.01971428,  0.0722929],
  [ 0.1248705,   0.17744702,  0.23002243,  0.28259667,  0.33516969],
  [ 0.38774145,  0.44031188,  0.49288093,  0.54544852,  0.59801459]])
expected_v = np.asarray([
  [ 0.69966,     0.68908382,  0.67851319,  0.66794809,  0.65738853,],
  [ 0.64683452,  0.63628604,  0.6257431,   0.61520571,  0.60467385,],
  [ 0.59414753,  0.58362676,  0.57311152,  0.56260183,  0.55209767,],
  [ 0.54159906,  0.53110598,  0.52061845,  0.51013645,  0.49966,   ]])
expected_m = np.asarray([
  [ 0.48,        0.49947368,  0.51894737,  0.53842105,  0.55789474],
  [ 0.57736842,  0.59684211,  0.61631579,  0.63578947,  0.65526316],
  [ 0.67473684,  0.69421053,  0.71368421,  0.73315789,  0.75263158],
  [ 0.77210526,  0.79157895,  0.81105263,  0.83052632,  0.85      ]])

# You should see relative errors around e-7 or less
print('next_w error: ', rel_error(expected_next_w, next_w))
print('v error: ', rel_error(expected_v, config['v']))
print('m error: ', rel_error(expected_m, config['m']))
```

```
next_w error:  1.1395691798535431e-07
v error:  4.208314038113071e-09
m error:  4.214963193114416e-09
```

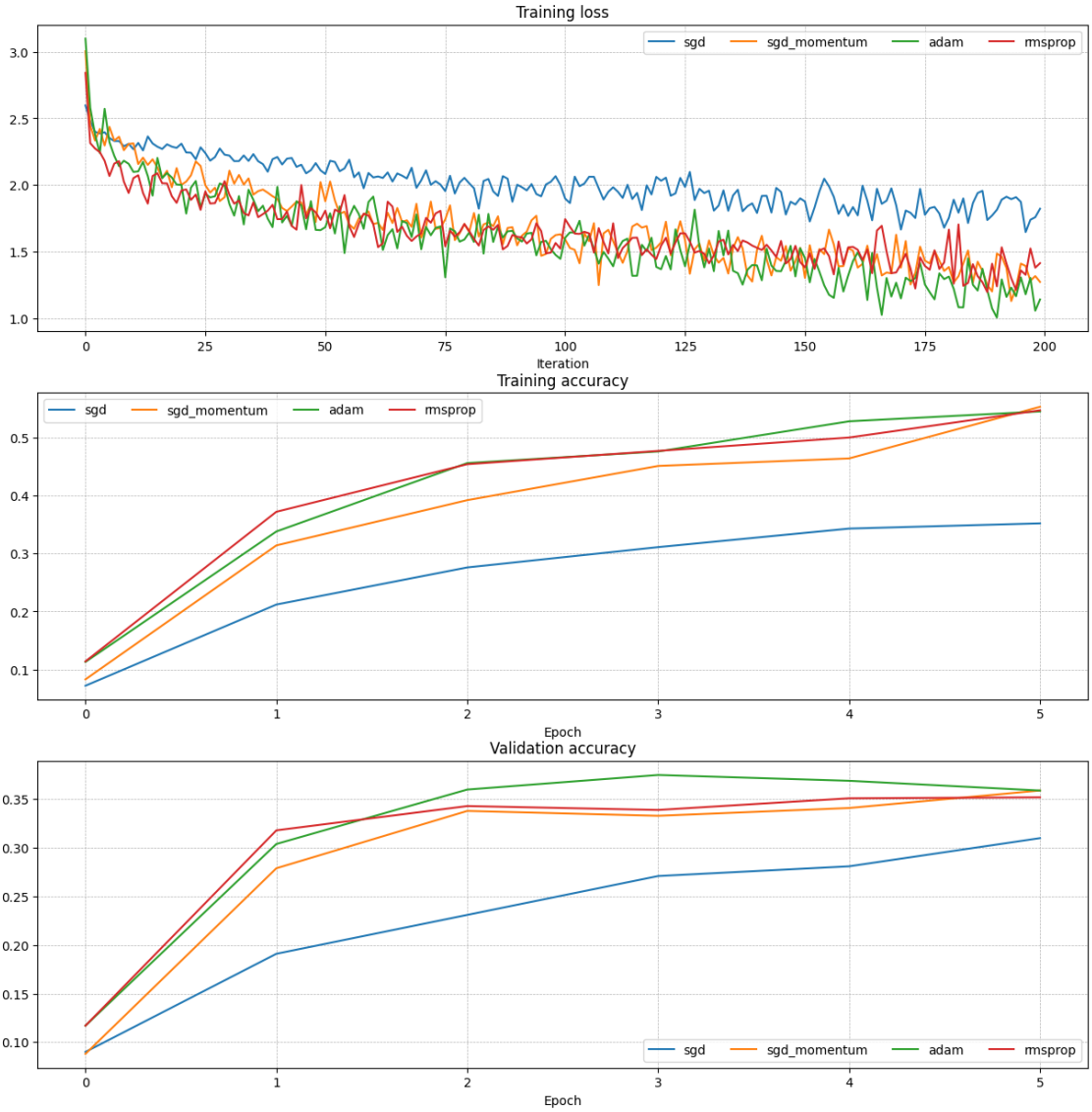
Once you have debugged your RMSProp and Adam implementations, run the following to train a pair of deep networks using these new update rules:

[81]:
```
learning_rates = {'rmsprop': 1e-4, 'adam': 1e-3}
for update_rule in ['adam', 'rmsprop']:
    print('Running with ', update_rule)
    model = FullyConnectedNet(
        [100, 100, 100, 100, 100],
        weight_scale=5e-2
    )
    solver = Solver(
```

```
        model,
        small_data,
        num_epochs=5,
        batch_size=100,
        update_rule=update_rule,
        optim_config={'learning_rate': learning_rates[update_rule]},
        verbose=True
    )
    solvers[update_rule] = solver
    solver.train()
    print()

fig, axes = plt.subplots(3, 1, figsize=(15, 15))

axes[0].set_title('Training loss')
axes[0].set_xlabel('Iteration')
axes[1].set_title('Training accuracy')
axes[1].set_xlabel('Epoch')
axes[2].set_title('Validation accuracy')
axes[2].set_xlabel('Epoch')

for update_rule, solver in solvers.items():
    axes[0].plot(solver.loss_history, label=f"{update_rule}")
    axes[1].plot(solver.train_acc_history, label=f"{update_rule}")
    axes[2].plot(solver.val_acc_history, label=f"{update_rule}")

for ax in axes:
    ax.legend(loc='best', ncol=4)
    ax.grid(linestyle='--', linewidth=0.5)

plt.show()
```

```
Running with  adam
(Iteration 1 / 200) loss: 3.098972
(Epoch 0 / 5) train acc: 0.113000; val_acc: 0.117000
(Iteration 11 / 200) loss: 2.097885
(Iteration 21 / 200) loss: 2.000393
(Iteration 31 / 200) loss: 1.856791
(Epoch 1 / 5) train acc: 0.338000; val_acc: 0.304000
(Iteration 41 / 200) loss: 1.986063
(Iteration 51 / 200) loss: 1.682858
(Iteration 61 / 200) loss: 1.912636
(Iteration 71 / 200) loss: 1.516613
(Epoch 2 / 5) train acc: 0.456000; val_acc: 0.360000
(Iteration 81 / 200) loss: 1.642533
(Iteration 91 / 200) loss: 1.598813
(Iteration 101 / 200) loss: 1.603976
```

```
(Iteration 111 / 200) loss: 1.388661
(Epoch 3 / 5) train acc: 0.476000; val_acc: 0.375000
(Iteration 121 / 200) loss: 1.366577
(Iteration 131 / 200) loss: 1.523444
(Iteration 141 / 200) loss: 1.395131
(Iteration 151 / 200) loss: 1.449838
(Epoch 4 / 5) train acc: 0.528000; val_acc: 0.369000
(Iteration 161 / 200) loss: 1.427463
(Iteration 171 / 200) loss: 1.147547
(Iteration 181 / 200) loss: 1.311655
(Iteration 191 / 200) loss: 1.003650
(Epoch 5 / 5) train acc: 0.545000; val_acc: 0.359000

Running with  rmsprop
(Iteration 1 / 200) loss: 2.841640
(Epoch 0 / 5) train acc: 0.114000; val_acc: 0.117000
(Iteration 11 / 200) loss: 2.051337
(Iteration 21 / 200) loss: 1.958493
(Iteration 31 / 200) loss: 1.926533
(Epoch 1 / 5) train acc: 0.372000; val_acc: 0.318000
(Iteration 41 / 200) loss: 1.742578
(Iteration 51 / 200) loss: 1.806655
(Iteration 61 / 200) loss: 1.705702
(Iteration 71 / 200) loss: 1.636658
(Epoch 2 / 5) train acc: 0.454000; val_acc: 0.343000
(Iteration 81 / 200) loss: 1.654176
(Iteration 91 / 200) loss: 1.610795
(Iteration 101 / 200) loss: 1.743135
(Iteration 111 / 200) loss: 1.711719
(Epoch 3 / 5) train acc: 0.477000; val_acc: 0.339000
(Iteration 121 / 200) loss: 1.543767
(Iteration 131 / 200) loss: 1.411418
(Iteration 141 / 200) loss: 1.528201
(Iteration 151 / 200) loss: 1.379760
(Epoch 4 / 5) train acc: 0.500000; val_acc: 0.351000
(Iteration 161 / 200) loss: 1.534842
(Iteration 171 / 200) loss: 1.405442
(Iteration 181 / 200) loss: 1.664724
(Iteration 191 / 200) loss: 1.238276
(Epoch 5 / 5) train acc: 0.547000; val_acc: 0.352000
```



2.3 Inline Question 2:

AdaGrad, like Adam, is a per-parameter optimization method that uses the following update rule:

```
cache += dw**2
w += - learning_rate * dw / (np.sqrt(cache) + eps)
```

John notices that when he was training a network with AdaGrad that the updates became very small, and that his network was learning slowly. Using your knowledge of the AdaGrad update rule, why do you think the updates would become very small? Would Adam have the same issue?

2.4 Answer:

Since the cache is cumulative, it will always grow bigger than before. Therefore, the denominator of the expression on the second line will become bigger and bigger, resulting in the step size decaying to zero.

3 Train a Good Model!

Train the best fully connected model that you can on CIFAR-10, storing your best model in the `best_model` variable. We require you to get at least 50% accuracy on the validation set using a fully connected network.

[101]:
```
best_model = None

################################################################################
# TODO: Train the best FullyConnectedNet that you can on CIFAR-10. Store your␣
 ↪best model in  #
# the best_model variable.                                                     #
################################################################################
# *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****

solvers = {}
num_train = 10000
new_data = {
  'X_train': data['X_train'][:num_train],
  'y_train': data['y_train'][:num_train],
  'X_val': data['X_val'],
  'y_val': data['y_val'],
}
learning_rates = [1e-3, 5e-4]
update_rules = ['rmsprop', 'adam']
weight_scale = [1e-2, 5e-2]
best_acc = -1

for update_rule in update_rules:
  for lr in learning_rates:
    for weight in weight_scale:
      model = FullyConnectedNet(
              [100, 100, 100, 100, 100],
              weight_scale=weight
          )

      solver = Solver(
          model,
          data,
          num_epochs=10,
          batch_size=100,
          update_rule=update_rule,
```

```
          optim_config={'learning_rate': lr},
          verbose=False
      )
      solvers[update_rule] = solver
      solver.train()

      if solver.best_val_acc > best_acc:
        best_acc = solver.best_val_acc
        best_model = model

print("best accuarcy:" , best_acc)
# *****END OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
################################################################################
#                              END OF YOUR CODE                                #
################################################################################
```

```
(Iteration 1 / 4900) loss: 2.302588
(Epoch 0 / 10) train acc: 0.117000; val_acc: 0.113000
(Iteration 11 / 4900) loss: 2.274554
(Iteration 21 / 4900) loss: 2.093774
(Iteration 31 / 4900) loss: 2.037357
(Iteration 41 / 4900) loss: 2.164575
(Iteration 51 / 4900) loss: 2.004596
(Iteration 61 / 4900) loss: 2.061790
(Iteration 71 / 4900) loss: 1.948057
(Iteration 81 / 4900) loss: 1.786122
(Iteration 91 / 4900) loss: 1.995147
(Iteration 101 / 4900) loss: 1.837101
(Iteration 111 / 4900) loss: 1.959208
(Iteration 121 / 4900) loss: 1.900710
(Iteration 131 / 4900) loss: 1.826006
(Iteration 141 / 4900) loss: 1.866170
(Iteration 151 / 4900) loss: 1.707058
(Iteration 161 / 4900) loss: 1.858022
(Iteration 171 / 4900) loss: 1.729564
(Iteration 181 / 4900) loss: 1.753774
(Iteration 191 / 4900) loss: 1.653800
(Iteration 201 / 4900) loss: 1.699336
(Iteration 211 / 4900) loss: 1.699998
(Iteration 221 / 4900) loss: 1.753145
(Iteration 231 / 4900) loss: 1.712812
(Iteration 241 / 4900) loss: 1.667963
(Iteration 251 / 4900) loss: 1.773786
(Iteration 261 / 4900) loss: 1.823099
(Iteration 271 / 4900) loss: 1.682273
(Iteration 281 / 4900) loss: 1.597856
(Iteration 291 / 4900) loss: 1.882703
```

```
(Iteration 301 / 4900) loss: 1.693356
(Iteration 311 / 4900) loss: 1.668815
(Iteration 321 / 4900) loss: 1.794598
(Iteration 331 / 4900) loss: 1.659837
(Iteration 341 / 4900) loss: 1.805288
(Iteration 351 / 4900) loss: 1.474176
(Iteration 361 / 4900) loss: 1.613831
(Iteration 371 / 4900) loss: 1.794556
(Iteration 381 / 4900) loss: 1.609809
(Iteration 391 / 4900) loss: 1.574716
(Iteration 401 / 4900) loss: 1.420814
(Iteration 411 / 4900) loss: 1.741513
(Iteration 421 / 4900) loss: 1.680268
(Iteration 431 / 4900) loss: 1.576518
(Iteration 441 / 4900) loss: 1.653658
(Iteration 451 / 4900) loss: 1.692635
(Iteration 461 / 4900) loss: 1.731404
(Iteration 471 / 4900) loss: 1.727852
(Iteration 481 / 4900) loss: 1.399351
(Epoch 1 / 10) train acc: 0.391000; val_acc: 0.421000
(Iteration 491 / 4900) loss: 1.641618
(Iteration 501 / 4900) loss: 1.560256
(Iteration 511 / 4900) loss: 1.628744
(Iteration 521 / 4900) loss: 1.594800
(Iteration 531 / 4900) loss: 1.571290
(Iteration 541 / 4900) loss: 1.429431
(Iteration 551 / 4900) loss: 1.567375
(Iteration 561 / 4900) loss: 1.693338
(Iteration 571 / 4900) loss: 1.603749
(Iteration 581 / 4900) loss: 1.573313
(Iteration 591 / 4900) loss: 1.297412
(Iteration 601 / 4900) loss: 1.644284
(Iteration 611 / 4900) loss: 1.398804
(Iteration 621 / 4900) loss: 1.673908
(Iteration 631 / 4900) loss: 1.620067
(Iteration 641 / 4900) loss: 1.737212
(Iteration 651 / 4900) loss: 1.431884
(Iteration 661 / 4900) loss: 1.633106
(Iteration 671 / 4900) loss: 1.490354
(Iteration 681 / 4900) loss: 1.481226
(Iteration 691 / 4900) loss: 1.662829
(Iteration 701 / 4900) loss: 1.580340
(Iteration 711 / 4900) loss: 1.527230
(Iteration 721 / 4900) loss: 1.640131
(Iteration 731 / 4900) loss: 1.672569
(Iteration 741 / 4900) loss: 1.490022
(Iteration 751 / 4900) loss: 1.539347
(Iteration 761 / 4900) loss: 1.498847
```

```
(Iteration 771 / 4900) loss: 1.461256
(Iteration 781 / 4900) loss: 1.491611
(Iteration 791 / 4900) loss: 1.591719
(Iteration 801 / 4900) loss: 1.639210
(Iteration 811 / 4900) loss: 1.512793
(Iteration 821 / 4900) loss: 1.414699
(Iteration 831 / 4900) loss: 1.436372
(Iteration 841 / 4900) loss: 1.490814
(Iteration 851 / 4900) loss: 1.522637
(Iteration 861 / 4900) loss: 1.578699
(Iteration 871 / 4900) loss: 1.515876
(Iteration 881 / 4900) loss: 1.532744
(Iteration 891 / 4900) loss: 1.626903
(Iteration 901 / 4900) loss: 1.469055
(Iteration 911 / 4900) loss: 1.281128
(Iteration 921 / 4900) loss: 1.691751
(Iteration 931 / 4900) loss: 1.538592
(Iteration 941 / 4900) loss: 1.487989
(Iteration 951 / 4900) loss: 1.673859
(Iteration 961 / 4900) loss: 1.548315
(Iteration 971 / 4900) loss: 1.569839
(Epoch 2 / 10) train acc: 0.465000; val_acc: 0.478000
(Iteration 981 / 4900) loss: 1.480700
(Iteration 991 / 4900) loss: 1.665412
(Iteration 1001 / 4900) loss: 1.505870
(Iteration 1011 / 4900) loss: 1.472279
(Iteration 1021 / 4900) loss: 1.448823
(Iteration 1031 / 4900) loss: 1.473723
(Iteration 1041 / 4900) loss: 1.759517
(Iteration 1051 / 4900) loss: 1.431417
(Iteration 1061 / 4900) loss: 1.374176
(Iteration 1071 / 4900) loss: 1.530680
(Iteration 1081 / 4900) loss: 1.623797
(Iteration 1091 / 4900) loss: 1.544257
(Iteration 1101 / 4900) loss: 1.483414
(Iteration 1111 / 4900) loss: 1.536587
(Iteration 1121 / 4900) loss: 1.416721
(Iteration 1131 / 4900) loss: 1.622737
(Iteration 1141 / 4900) loss: 1.472786
(Iteration 1151 / 4900) loss: 1.513017
(Iteration 1161 / 4900) loss: 1.600422
(Iteration 1171 / 4900) loss: 1.266403
(Iteration 1181 / 4900) loss: 1.594087
(Iteration 1191 / 4900) loss: 1.479286
(Iteration 1201 / 4900) loss: 1.486352
(Iteration 1211 / 4900) loss: 1.706366
(Iteration 1221 / 4900) loss: 1.555302
(Iteration 1231 / 4900) loss: 1.495104
```

```
(Iteration 1241 / 4900) loss: 1.547075
(Iteration 1251 / 4900) loss: 1.409607
(Iteration 1261 / 4900) loss: 1.506105
(Iteration 1271 / 4900) loss: 1.422578
(Iteration 1281 / 4900) loss: 1.503162
(Iteration 1291 / 4900) loss: 1.447601
(Iteration 1301 / 4900) loss: 1.489859
(Iteration 1311 / 4900) loss: 1.468413
(Iteration 1321 / 4900) loss: 1.393979
(Iteration 1331 / 4900) loss: 1.257231
(Iteration 1341 / 4900) loss: 1.371219
(Iteration 1351 / 4900) loss: 1.366158
(Iteration 1361 / 4900) loss: 1.424818
(Iteration 1371 / 4900) loss: 1.462059
(Iteration 1381 / 4900) loss: 1.578927
(Iteration 1391 / 4900) loss: 1.524318
(Iteration 1401 / 4900) loss: 1.456042
(Iteration 1411 / 4900) loss: 1.437131
(Iteration 1421 / 4900) loss: 1.483197
(Iteration 1431 / 4900) loss: 1.293999
(Iteration 1441 / 4900) loss: 1.389933
(Iteration 1451 / 4900) loss: 1.396825
(Iteration 1461 / 4900) loss: 1.387243
(Epoch 3 / 10) train acc: 0.499000; val_acc: 0.477000
(Iteration 1471 / 4900) loss: 1.302426
(Iteration 1481 / 4900) loss: 1.323538
(Iteration 1491 / 4900) loss: 1.626021
(Iteration 1501 / 4900) loss: 1.300591
(Iteration 1511 / 4900) loss: 1.350660
(Iteration 1521 / 4900) loss: 1.343345
(Iteration 1531 / 4900) loss: 1.403142
(Iteration 1541 / 4900) loss: 1.250931
(Iteration 1551 / 4900) loss: 1.344214
(Iteration 1561 / 4900) loss: 1.414029
(Iteration 1571 / 4900) loss: 1.614846
(Iteration 1581 / 4900) loss: 1.485269
(Iteration 1591 / 4900) loss: 1.294082
(Iteration 1601 / 4900) loss: 1.563735
(Iteration 1611 / 4900) loss: 1.360622
(Iteration 1621 / 4900) loss: 1.344904
(Iteration 1631 / 4900) loss: 1.455902
(Iteration 1641 / 4900) loss: 1.378357
(Iteration 1651 / 4900) loss: 1.256155
(Iteration 1661 / 4900) loss: 1.277330
(Iteration 1671 / 4900) loss: 1.455878
(Iteration 1681 / 4900) loss: 1.344612
(Iteration 1691 / 4900) loss: 1.379477
(Iteration 1701 / 4900) loss: 1.341902
```

```
(Iteration 1711 / 4900) loss: 1.453547
(Iteration 1721 / 4900) loss: 1.423862
(Iteration 1731 / 4900) loss: 1.438645
(Iteration 1741 / 4900) loss: 1.286816
(Iteration 1751 / 4900) loss: 1.139452
(Iteration 1761 / 4900) loss: 1.369791
(Iteration 1771 / 4900) loss: 1.325428
(Iteration 1781 / 4900) loss: 1.280357
(Iteration 1791 / 4900) loss: 1.413694
(Iteration 1801 / 4900) loss: 1.349822
(Iteration 1811 / 4900) loss: 1.341317
(Iteration 1821 / 4900) loss: 1.440910
(Iteration 1831 / 4900) loss: 1.260025
(Iteration 1841 / 4900) loss: 1.300002
(Iteration 1851 / 4900) loss: 1.321822
(Iteration 1861 / 4900) loss: 1.186451
(Iteration 1871 / 4900) loss: 1.498725
(Iteration 1881 / 4900) loss: 1.233876
(Iteration 1891 / 4900) loss: 1.385809
(Iteration 1901 / 4900) loss: 1.346101
(Iteration 1911 / 4900) loss: 1.376342
(Iteration 1921 / 4900) loss: 1.441700
(Iteration 1931 / 4900) loss: 1.398471
(Iteration 1941 / 4900) loss: 1.221352
(Iteration 1951 / 4900) loss: 1.316821
(Epoch 4 / 10) train acc: 0.532000; val_acc: 0.503000
(Iteration 1961 / 4900) loss: 1.179257
(Iteration 1971 / 4900) loss: 1.241447
(Iteration 1981 / 4900) loss: 1.233069
(Iteration 1991 / 4900) loss: 1.443147
(Iteration 2001 / 4900) loss: 1.370708
(Iteration 2011 / 4900) loss: 1.294862
(Iteration 2021 / 4900) loss: 1.351539
(Iteration 2031 / 4900) loss: 1.393662
(Iteration 2041 / 4900) loss: 1.418025
(Iteration 2051 / 4900) loss: 1.420902
(Iteration 2061 / 4900) loss: 1.305663
(Iteration 2071 / 4900) loss: 1.400353
(Iteration 2081 / 4900) loss: 1.485451
(Iteration 2091 / 4900) loss: 1.191736
(Iteration 2101 / 4900) loss: 1.425526
(Iteration 2111 / 4900) loss: 1.326067
(Iteration 2121 / 4900) loss: 1.563482
(Iteration 2131 / 4900) loss: 1.426311
(Iteration 2141 / 4900) loss: 1.429389
(Iteration 2151 / 4900) loss: 1.340769
(Iteration 2161 / 4900) loss: 1.434906
(Iteration 2171 / 4900) loss: 1.275127
```

```
(Iteration 2181 / 4900) loss: 1.289171
(Iteration 2191 / 4900) loss: 1.498220
(Iteration 2201 / 4900) loss: 1.371377
(Iteration 2211 / 4900) loss: 1.163765
(Iteration 2221 / 4900) loss: 1.319068
(Iteration 2231 / 4900) loss: 1.471849
(Iteration 2241 / 4900) loss: 1.301774
(Iteration 2251 / 4900) loss: 1.431743
(Iteration 2261 / 4900) loss: 1.405814
(Iteration 2271 / 4900) loss: 1.333293
(Iteration 2281 / 4900) loss: 1.357637
(Iteration 2291 / 4900) loss: 1.322132
(Iteration 2301 / 4900) loss: 1.474354
(Iteration 2311 / 4900) loss: 1.400508
(Iteration 2321 / 4900) loss: 1.407934
(Iteration 2331 / 4900) loss: 1.327541
(Iteration 2341 / 4900) loss: 1.264041
(Iteration 2351 / 4900) loss: 1.399587
(Iteration 2361 / 4900) loss: 1.236175
(Iteration 2371 / 4900) loss: 1.316474
(Iteration 2381 / 4900) loss: 1.373889
(Iteration 2391 / 4900) loss: 1.459554
(Iteration 2401 / 4900) loss: 1.427878
(Iteration 2411 / 4900) loss: 1.326804
(Iteration 2421 / 4900) loss: 1.278358
(Iteration 2431 / 4900) loss: 1.344950
(Iteration 2441 / 4900) loss: 1.418731
(Epoch 5 / 10) train acc: 0.534000; val_acc: 0.483000
(Iteration 2451 / 4900) loss: 1.298034
(Iteration 2461 / 4900) loss: 1.385522
(Iteration 2471 / 4900) loss: 1.197822
(Iteration 2481 / 4900) loss: 1.474925
(Iteration 2491 / 4900) loss: 1.298277
(Iteration 2501 / 4900) loss: 1.125568
(Iteration 2511 / 4900) loss: 1.297616
(Iteration 2521 / 4900) loss: 1.356052
(Iteration 2531 / 4900) loss: 1.305076
(Iteration 2541 / 4900) loss: 1.277526
(Iteration 2551 / 4900) loss: 1.261010
(Iteration 2561 / 4900) loss: 1.268065
(Iteration 2571 / 4900) loss: 1.412475
(Iteration 2581 / 4900) loss: 1.283386
(Iteration 2591 / 4900) loss: 1.493452
(Iteration 2601 / 4900) loss: 1.380713
(Iteration 2611 / 4900) loss: 1.182958
(Iteration 2621 / 4900) loss: 1.454405
(Iteration 2631 / 4900) loss: 1.224819
(Iteration 2641 / 4900) loss: 1.211273
```

```
(Iteration 2651 / 4900) loss: 1.447837
(Iteration 2661 / 4900) loss: 1.186090
(Iteration 2671 / 4900) loss: 1.438507
(Iteration 2681 / 4900) loss: 1.298572
(Iteration 2691 / 4900) loss: 1.255041
(Iteration 2701 / 4900) loss: 1.172362
(Iteration 2711 / 4900) loss: 1.087099
(Iteration 2721 / 4900) loss: 1.329450
(Iteration 2731 / 4900) loss: 1.238312
(Iteration 2741 / 4900) loss: 1.238522
(Iteration 2751 / 4900) loss: 1.371475
(Iteration 2761 / 4900) loss: 1.190561
(Iteration 2771 / 4900) loss: 1.433965
(Iteration 2781 / 4900) loss: 1.369579
(Iteration 2791 / 4900) loss: 1.145263
(Iteration 2801 / 4900) loss: 1.265798
(Iteration 2811 / 4900) loss: 1.422659
(Iteration 2821 / 4900) loss: 1.350592
(Iteration 2831 / 4900) loss: 1.202631
(Iteration 2841 / 4900) loss: 1.171261
(Iteration 2851 / 4900) loss: 1.365547
(Iteration 2861 / 4900) loss: 1.109905
(Iteration 2871 / 4900) loss: 1.411781
(Iteration 2881 / 4900) loss: 1.344014
(Iteration 2891 / 4900) loss: 1.043752
(Iteration 2901 / 4900) loss: 1.090482
(Iteration 2911 / 4900) loss: 1.309434
(Iteration 2921 / 4900) loss: 1.260439
(Iteration 2931 / 4900) loss: 1.366757
(Epoch 6 / 10) train acc: 0.540000; val_acc: 0.490000
(Iteration 2941 / 4900) loss: 1.293799
(Iteration 2951 / 4900) loss: 1.170398
(Iteration 2961 / 4900) loss: 1.436437
(Iteration 2971 / 4900) loss: 1.412139
(Iteration 2981 / 4900) loss: 1.402499
(Iteration 2991 / 4900) loss: 1.366789
(Iteration 3001 / 4900) loss: 1.368475
(Iteration 3011 / 4900) loss: 1.358581
(Iteration 3021 / 4900) loss: 1.190087
(Iteration 3031 / 4900) loss: 1.187673
(Iteration 3041 / 4900) loss: 1.391039
(Iteration 3051 / 4900) loss: 1.228707
(Iteration 3061 / 4900) loss: 1.248515
(Iteration 3071 / 4900) loss: 1.309771
(Iteration 3081 / 4900) loss: 1.368130
(Iteration 3091 / 4900) loss: 1.312742
(Iteration 3101 / 4900) loss: 1.210600
(Iteration 3111 / 4900) loss: 1.079641
```

```
(Iteration 3121 / 4900) loss: 1.204574
(Iteration 3131 / 4900) loss: 1.275766
(Iteration 3141 / 4900) loss: 1.363920
(Iteration 3151 / 4900) loss: 1.235023
(Iteration 3161 / 4900) loss: 1.123736
(Iteration 3171 / 4900) loss: 1.229765
(Iteration 3181 / 4900) loss: 1.207723
(Iteration 3191 / 4900) loss: 1.237756
(Iteration 3201 / 4900) loss: 1.065068
(Iteration 3211 / 4900) loss: 1.164025
(Iteration 3221 / 4900) loss: 1.238880
(Iteration 3231 / 4900) loss: 1.243384
(Iteration 3241 / 4900) loss: 1.337556
(Iteration 3251 / 4900) loss: 1.396762
(Iteration 3261 / 4900) loss: 1.260859
(Iteration 3271 / 4900) loss: 1.384657
(Iteration 3281 / 4900) loss: 1.397913
(Iteration 3291 / 4900) loss: 1.236805
(Iteration 3301 / 4900) loss: 1.240583
(Iteration 3311 / 4900) loss: 1.385644
(Iteration 3321 / 4900) loss: 1.107971
(Iteration 3331 / 4900) loss: 1.151583
(Iteration 3341 / 4900) loss: 1.303691
(Iteration 3351 / 4900) loss: 1.355323
(Iteration 3361 / 4900) loss: 1.216162
(Iteration 3371 / 4900) loss: 1.179747
(Iteration 3381 / 4900) loss: 1.336754
(Iteration 3391 / 4900) loss: 1.226285
(Iteration 3401 / 4900) loss: 1.312962
(Iteration 3411 / 4900) loss: 1.187140
(Iteration 3421 / 4900) loss: 1.254617
(Epoch 7 / 10) train acc: 0.538000; val_acc: 0.511000
(Iteration 3431 / 4900) loss: 1.221424
(Iteration 3441 / 4900) loss: 1.198421
(Iteration 3451 / 4900) loss: 1.264274
(Iteration 3461 / 4900) loss: 1.134505
(Iteration 3471 / 4900) loss: 1.334583
(Iteration 3481 / 4900) loss: 1.231853
(Iteration 3491 / 4900) loss: 1.179839
(Iteration 3501 / 4900) loss: 1.555924
(Iteration 3511 / 4900) loss: 1.298293
(Iteration 3521 / 4900) loss: 1.469079
(Iteration 3531 / 4900) loss: 1.038743
(Iteration 3541 / 4900) loss: 1.138459
(Iteration 3551 / 4900) loss: 1.097565
(Iteration 3561 / 4900) loss: 1.205278
(Iteration 3571 / 4900) loss: 1.105371
(Iteration 3581 / 4900) loss: 1.338106
```

```
(Iteration 3591 / 4900) loss: 1.212763
(Iteration 3601 / 4900) loss: 1.232527
(Iteration 3611 / 4900) loss: 1.294980
(Iteration 3621 / 4900) loss: 1.346015
(Iteration 3631 / 4900) loss: 1.386865
(Iteration 3641 / 4900) loss: 1.194419
(Iteration 3651 / 4900) loss: 1.226885
(Iteration 3661 / 4900) loss: 1.311240
(Iteration 3671 / 4900) loss: 1.219662
(Iteration 3681 / 4900) loss: 1.273994
(Iteration 3691 / 4900) loss: 1.283440
(Iteration 3701 / 4900) loss: 1.123454
(Iteration 3711 / 4900) loss: 1.044706
(Iteration 3721 / 4900) loss: 1.355108
(Iteration 3731 / 4900) loss: 1.220314
(Iteration 3741 / 4900) loss: 1.144807
(Iteration 3751 / 4900) loss: 1.233819
(Iteration 3761 / 4900) loss: 1.289083
(Iteration 3771 / 4900) loss: 1.327851
(Iteration 3781 / 4900) loss: 1.342838
(Iteration 3791 / 4900) loss: 1.212060
(Iteration 3801 / 4900) loss: 1.047724
(Iteration 3811 / 4900) loss: 1.159458
(Iteration 3821 / 4900) loss: 1.149918
(Iteration 3831 / 4900) loss: 1.219109
(Iteration 3841 / 4900) loss: 1.510892
(Iteration 3851 / 4900) loss: 1.210002
(Iteration 3861 / 4900) loss: 1.077856
(Iteration 3871 / 4900) loss: 1.325773
(Iteration 3881 / 4900) loss: 1.149388
(Iteration 3891 / 4900) loss: 1.145899
(Iteration 3901 / 4900) loss: 1.227306
(Iteration 3911 / 4900) loss: 1.362342
(Epoch 8 / 10) train acc: 0.576000; val_acc: 0.500000
(Iteration 3921 / 4900) loss: 1.194385
(Iteration 3931 / 4900) loss: 1.152486
(Iteration 3941 / 4900) loss: 1.169495
(Iteration 3951 / 4900) loss: 1.241639
(Iteration 3961 / 4900) loss: 1.327793
(Iteration 3971 / 4900) loss: 1.182512
(Iteration 3981 / 4900) loss: 1.296830
(Iteration 3991 / 4900) loss: 1.284003
(Iteration 4001 / 4900) loss: 1.273145
(Iteration 4011 / 4900) loss: 1.198966
(Iteration 4021 / 4900) loss: 1.207876
(Iteration 4031 / 4900) loss: 1.193985
(Iteration 4041 / 4900) loss: 1.211467
(Iteration 4051 / 4900) loss: 1.085605
```

```
(Iteration 4061 / 4900) loss: 1.313495
(Iteration 4071 / 4900) loss: 0.897902
(Iteration 4081 / 4900) loss: 1.032897
(Iteration 4091 / 4900) loss: 1.005766
(Iteration 4101 / 4900) loss: 1.147251
(Iteration 4111 / 4900) loss: 1.171045
(Iteration 4121 / 4900) loss: 1.015959
(Iteration 4131 / 4900) loss: 1.296104
(Iteration 4141 / 4900) loss: 1.046685
(Iteration 4151 / 4900) loss: 1.048045
(Iteration 4161 / 4900) loss: 1.115033
(Iteration 4171 / 4900) loss: 1.202775
(Iteration 4181 / 4900) loss: 1.286982
(Iteration 4191 / 4900) loss: 1.261008
(Iteration 4201 / 4900) loss: 1.107907
(Iteration 4211 / 4900) loss: 1.218034
(Iteration 4221 / 4900) loss: 1.064377
(Iteration 4231 / 4900) loss: 1.126232
(Iteration 4241 / 4900) loss: 0.984182
(Iteration 4251 / 4900) loss: 1.183905
(Iteration 4261 / 4900) loss: 1.383567
(Iteration 4271 / 4900) loss: 1.191938
(Iteration 4281 / 4900) loss: 1.196728
(Iteration 4291 / 4900) loss: 1.191410
(Iteration 4301 / 4900) loss: 1.127272
(Iteration 4311 / 4900) loss: 1.342345
(Iteration 4321 / 4900) loss: 1.064998
(Iteration 4331 / 4900) loss: 1.034291
(Iteration 4341 / 4900) loss: 0.976148
(Iteration 4351 / 4900) loss: 1.215657
(Iteration 4361 / 4900) loss: 1.344030
(Iteration 4371 / 4900) loss: 1.417407
(Iteration 4381 / 4900) loss: 1.141638
(Iteration 4391 / 4900) loss: 1.424686
(Iteration 4401 / 4900) loss: 1.213263
(Epoch 9 / 10) train acc: 0.592000; val_acc: 0.502000
(Iteration 4411 / 4900) loss: 1.350079
(Iteration 4421 / 4900) loss: 1.231165
(Iteration 4431 / 4900) loss: 1.189024
(Iteration 4441 / 4900) loss: 1.255646
(Iteration 4451 / 4900) loss: 1.074281
(Iteration 4461 / 4900) loss: 1.094807
(Iteration 4471 / 4900) loss: 1.296822
(Iteration 4481 / 4900) loss: 1.364974
(Iteration 4491 / 4900) loss: 0.975501
(Iteration 4501 / 4900) loss: 1.054252
(Iteration 4511 / 4900) loss: 1.324758
(Iteration 4521 / 4900) loss: 1.131737
```

```
(Iteration 4531 / 4900) loss: 0.956719
(Iteration 4541 / 4900) loss: 1.120653
(Iteration 4551 / 4900) loss: 1.229551
(Iteration 4561 / 4900) loss: 1.247129
(Iteration 4571 / 4900) loss: 1.292075
(Iteration 4581 / 4900) loss: 1.121825
(Iteration 4591 / 4900) loss: 1.264085
(Iteration 4601 / 4900) loss: 1.277648
(Iteration 4611 / 4900) loss: 1.286035
(Iteration 4621 / 4900) loss: 1.158145
(Iteration 4631 / 4900) loss: 1.217716
(Iteration 4641 / 4900) loss: 1.015386
(Iteration 4651 / 4900) loss: 1.405223
(Iteration 4661 / 4900) loss: 1.028716
(Iteration 4671 / 4900) loss: 1.111961
(Iteration 4681 / 4900) loss: 1.215284
(Iteration 4691 / 4900) loss: 1.185476
(Iteration 4701 / 4900) loss: 1.073776
(Iteration 4711 / 4900) loss: 0.968309
(Iteration 4721 / 4900) loss: 1.303335
(Iteration 4731 / 4900) loss: 1.237516
(Iteration 4741 / 4900) loss: 1.216763
(Iteration 4751 / 4900) loss: 1.314637
(Iteration 4761 / 4900) loss: 1.169969
(Iteration 4771 / 4900) loss: 1.172817
(Iteration 4781 / 4900) loss: 1.024308
(Iteration 4791 / 4900) loss: 1.105730
(Iteration 4801 / 4900) loss: 1.091022
(Iteration 4811 / 4900) loss: 1.145693
(Iteration 4821 / 4900) loss: 1.271722
(Iteration 4831 / 4900) loss: 1.221824
(Iteration 4841 / 4900) loss: 1.125832
(Iteration 4851 / 4900) loss: 1.136847
(Iteration 4861 / 4900) loss: 1.385332
(Iteration 4871 / 4900) loss: 1.164342
(Iteration 4881 / 4900) loss: 1.104651
(Iteration 4891 / 4900) loss: 1.266159
(Epoch 10 / 10) train acc: 0.600000; val_acc: 0.507000
(Iteration 1 / 4900) loss: 3.152001
(Epoch 0 / 10) train acc: 0.114000; val_acc: 0.108000
(Iteration 11 / 4900) loss: 2.271947
(Iteration 21 / 4900) loss: 2.046200
(Iteration 31 / 4900) loss: 2.073834
(Iteration 41 / 4900) loss: 1.978854
(Iteration 51 / 4900) loss: 2.030019
(Iteration 61 / 4900) loss: 1.925806
(Iteration 71 / 4900) loss: 1.915248
(Iteration 81 / 4900) loss: 1.765315
```

```
(Iteration 91 / 4900) loss: 2.030776
(Iteration 101 / 4900) loss: 1.791659
(Iteration 111 / 4900) loss: 1.820609
(Iteration 121 / 4900) loss: 1.688769
(Iteration 131 / 4900) loss: 1.786440
(Iteration 141 / 4900) loss: 1.804380
(Iteration 151 / 4900) loss: 1.806092
(Iteration 161 / 4900) loss: 1.566893
(Iteration 171 / 4900) loss: 1.849777
(Iteration 181 / 4900) loss: 1.730831
(Iteration 191 / 4900) loss: 1.774702
(Iteration 201 / 4900) loss: 1.661614
(Iteration 211 / 4900) loss: 1.813347
(Iteration 221 / 4900) loss: 1.731641
(Iteration 231 / 4900) loss: 1.798824
(Iteration 241 / 4900) loss: 1.751235
(Iteration 251 / 4900) loss: 1.836137
(Iteration 261 / 4900) loss: 1.700571
(Iteration 271 / 4900) loss: 1.646909
(Iteration 281 / 4900) loss: 1.613913
(Iteration 291 / 4900) loss: 1.704648
(Iteration 301 / 4900) loss: 1.653039
(Iteration 311 / 4900) loss: 1.504668
(Iteration 321 / 4900) loss: 1.441706
(Iteration 331 / 4900) loss: 1.832508
(Iteration 341 / 4900) loss: 1.741031
(Iteration 351 / 4900) loss: 1.783251
(Iteration 361 / 4900) loss: 1.501214
(Iteration 371 / 4900) loss: 1.733464
(Iteration 381 / 4900) loss: 1.641128
(Iteration 391 / 4900) loss: 1.620794
(Iteration 401 / 4900) loss: 1.669906
(Iteration 411 / 4900) loss: 1.673956
(Iteration 421 / 4900) loss: 1.510693
(Iteration 431 / 4900) loss: 1.718242
(Iteration 441 / 4900) loss: 1.618719
(Iteration 451 / 4900) loss: 1.664751
(Iteration 461 / 4900) loss: 1.526157
(Iteration 471 / 4900) loss: 1.551492
(Iteration 481 / 4900) loss: 1.565126
(Epoch 1 / 10) train acc: 0.409000; val_acc: 0.413000
(Iteration 491 / 4900) loss: 1.741740
(Iteration 501 / 4900) loss: 1.679282
(Iteration 511 / 4900) loss: 1.563694
(Iteration 521 / 4900) loss: 1.774669
(Iteration 531 / 4900) loss: 1.507369
(Iteration 541 / 4900) loss: 1.482503
(Iteration 551 / 4900) loss: 1.588616
```

```
(Iteration 561 / 4900) loss: 1.507773
(Iteration 571 / 4900) loss: 1.768496
(Iteration 581 / 4900) loss: 1.698492
(Iteration 591 / 4900) loss: 1.722356
(Iteration 601 / 4900) loss: 1.558556
(Iteration 611 / 4900) loss: 1.505957
(Iteration 621 / 4900) loss: 1.691826
(Iteration 631 / 4900) loss: 1.816483
(Iteration 641 / 4900) loss: 1.533315
(Iteration 651 / 4900) loss: 1.722940
(Iteration 661 / 4900) loss: 1.487221
(Iteration 671 / 4900) loss: 1.762317
(Iteration 681 / 4900) loss: 1.561188
(Iteration 691 / 4900) loss: 1.468675
(Iteration 701 / 4900) loss: 1.582845
(Iteration 711 / 4900) loss: 1.361024
(Iteration 721 / 4900) loss: 1.367550
(Iteration 731 / 4900) loss: 1.723015
(Iteration 741 / 4900) loss: 1.566057
(Iteration 751 / 4900) loss: 1.690368
(Iteration 761 / 4900) loss: 1.550958
(Iteration 771 / 4900) loss: 1.594769
(Iteration 781 / 4900) loss: 1.613354
(Iteration 791 / 4900) loss: 1.512395
(Iteration 801 / 4900) loss: 1.457475
(Iteration 811 / 4900) loss: 1.592837
(Iteration 821 / 4900) loss: 1.649506
(Iteration 831 / 4900) loss: 1.464881
(Iteration 841 / 4900) loss: 1.524961
(Iteration 851 / 4900) loss: 1.611328
(Iteration 861 / 4900) loss: 1.436822
(Iteration 871 / 4900) loss: 1.490180
(Iteration 881 / 4900) loss: 1.500707
(Iteration 891 / 4900) loss: 1.494306
(Iteration 901 / 4900) loss: 1.574042
(Iteration 911 / 4900) loss: 1.656150
(Iteration 921 / 4900) loss: 1.459347
(Iteration 931 / 4900) loss: 1.588498
(Iteration 941 / 4900) loss: 1.554416
(Iteration 951 / 4900) loss: 1.361095
(Iteration 961 / 4900) loss: 1.573548
(Iteration 971 / 4900) loss: 1.533014
(Epoch 2 / 10) train acc: 0.486000; val_acc: 0.450000
(Iteration 981 / 4900) loss: 1.523162
(Iteration 991 / 4900) loss: 1.477940
(Iteration 1001 / 4900) loss: 1.536763
(Iteration 1011 / 4900) loss: 1.400771
(Iteration 1021 / 4900) loss: 1.490980
```

```
(Iteration 1031 / 4900) loss: 1.471967
(Iteration 1041 / 4900) loss: 1.779970
(Iteration 1051 / 4900) loss: 1.386774
(Iteration 1061 / 4900) loss: 1.353790
(Iteration 1071 / 4900) loss: 1.540847
(Iteration 1081 / 4900) loss: 1.409257
(Iteration 1091 / 4900) loss: 1.487434
(Iteration 1101 / 4900) loss: 1.516287
(Iteration 1111 / 4900) loss: 1.413270
(Iteration 1121 / 4900) loss: 1.468332
(Iteration 1131 / 4900) loss: 1.484553
(Iteration 1141 / 4900) loss: 1.573251
(Iteration 1151 / 4900) loss: 1.337458
(Iteration 1161 / 4900) loss: 1.260654
(Iteration 1171 / 4900) loss: 1.475955
(Iteration 1181 / 4900) loss: 1.378127
(Iteration 1191 / 4900) loss: 1.328153
(Iteration 1201 / 4900) loss: 1.687396
(Iteration 1211 / 4900) loss: 1.523988
(Iteration 1221 / 4900) loss: 1.364829
(Iteration 1231 / 4900) loss: 1.652507
(Iteration 1241 / 4900) loss: 1.417813
(Iteration 1251 / 4900) loss: 1.431443
(Iteration 1261 / 4900) loss: 1.377001
(Iteration 1271 / 4900) loss: 1.526293
(Iteration 1281 / 4900) loss: 1.636678
(Iteration 1291 / 4900) loss: 1.329821
(Iteration 1301 / 4900) loss: 1.522117
(Iteration 1311 / 4900) loss: 1.278033
(Iteration 1321 / 4900) loss: 1.476433
(Iteration 1331 / 4900) loss: 1.391700
(Iteration 1341 / 4900) loss: 1.518769
(Iteration 1351 / 4900) loss: 1.519878
(Iteration 1361 / 4900) loss: 1.424584
(Iteration 1371 / 4900) loss: 1.258918
(Iteration 1381 / 4900) loss: 1.437267
(Iteration 1391 / 4900) loss: 1.334845
(Iteration 1401 / 4900) loss: 1.365111
(Iteration 1411 / 4900) loss: 1.449781
(Iteration 1421 / 4900) loss: 1.255986
(Iteration 1431 / 4900) loss: 1.416153
(Iteration 1441 / 4900) loss: 1.363972
(Iteration 1451 / 4900) loss: 1.690372
(Iteration 1461 / 4900) loss: 1.409279
(Epoch 3 / 10) train acc: 0.501000; val_acc: 0.465000
(Iteration 1471 / 4900) loss: 1.452430
(Iteration 1481 / 4900) loss: 1.396487
(Iteration 1491 / 4900) loss: 1.488403
```

```
(Iteration 1501 / 4900) loss: 1.436217
(Iteration 1511 / 4900) loss: 1.491240
(Iteration 1521 / 4900) loss: 1.366149
(Iteration 1531 / 4900) loss: 1.322744
(Iteration 1541 / 4900) loss: 1.554371
(Iteration 1551 / 4900) loss: 1.493204
(Iteration 1561 / 4900) loss: 1.501327
(Iteration 1571 / 4900) loss: 1.531351
(Iteration 1581 / 4900) loss: 1.553995
(Iteration 1591 / 4900) loss: 1.326578
(Iteration 1601 / 4900) loss: 1.551779
(Iteration 1611 / 4900) loss: 1.505065
(Iteration 1621 / 4900) loss: 1.409218
(Iteration 1631 / 4900) loss: 1.312146
(Iteration 1641 / 4900) loss: 1.031623
(Iteration 1651 / 4900) loss: 1.447945
(Iteration 1661 / 4900) loss: 1.285897
(Iteration 1671 / 4900) loss: 1.471199
(Iteration 1681 / 4900) loss: 1.416983
(Iteration 1691 / 4900) loss: 1.277293
(Iteration 1701 / 4900) loss: 1.385776
(Iteration 1711 / 4900) loss: 1.288589
(Iteration 1721 / 4900) loss: 1.527166
(Iteration 1731 / 4900) loss: 1.234939
(Iteration 1741 / 4900) loss: 1.757281
(Iteration 1751 / 4900) loss: 1.595070
(Iteration 1761 / 4900) loss: 1.301430
(Iteration 1771 / 4900) loss: 1.467054
(Iteration 1781 / 4900) loss: 1.232637
(Iteration 1791 / 4900) loss: 1.341963
(Iteration 1801 / 4900) loss: 1.338051
(Iteration 1811 / 4900) loss: 1.266226
(Iteration 1821 / 4900) loss: 1.172717
(Iteration 1831 / 4900) loss: 1.224195
(Iteration 1841 / 4900) loss: 1.342134
(Iteration 1851 / 4900) loss: 1.252948
(Iteration 1861 / 4900) loss: 1.406485
(Iteration 1871 / 4900) loss: 1.310679
(Iteration 1881 / 4900) loss: 1.366791
(Iteration 1891 / 4900) loss: 1.550808
(Iteration 1901 / 4900) loss: 1.409225
(Iteration 1911 / 4900) loss: 1.454775
(Iteration 1921 / 4900) loss: 1.387657
(Iteration 1931 / 4900) loss: 1.323893
(Iteration 1941 / 4900) loss: 1.511563
(Iteration 1951 / 4900) loss: 1.313106
(Epoch 4 / 10) train acc: 0.489000; val_acc: 0.456000
(Iteration 1961 / 4900) loss: 1.328332
```

```
(Iteration 1971 / 4900) loss: 1.448656
(Iteration 1981 / 4900) loss: 1.202493
(Iteration 1991 / 4900) loss: 1.433603
(Iteration 2001 / 4900) loss: 1.313200
(Iteration 2011 / 4900) loss: 1.380266
(Iteration 2021 / 4900) loss: 1.351057
(Iteration 2031 / 4900) loss: 1.286446
(Iteration 2041 / 4900) loss: 1.232725
(Iteration 2051 / 4900) loss: 1.116019
(Iteration 2061 / 4900) loss: 1.310271
(Iteration 2071 / 4900) loss: 1.302247
(Iteration 2081 / 4900) loss: 1.423316
(Iteration 2091 / 4900) loss: 1.446213
(Iteration 2101 / 4900) loss: 1.344448
(Iteration 2111 / 4900) loss: 1.305253
(Iteration 2121 / 4900) loss: 1.283549
(Iteration 2131 / 4900) loss: 1.357881
(Iteration 2141 / 4900) loss: 1.268685
(Iteration 2151 / 4900) loss: 1.208931
(Iteration 2161 / 4900) loss: 1.392395
(Iteration 2171 / 4900) loss: 1.344828
(Iteration 2181 / 4900) loss: 1.367081
(Iteration 2191 / 4900) loss: 1.327610
(Iteration 2201 / 4900) loss: 1.469025
(Iteration 2211 / 4900) loss: 1.483127
(Iteration 2221 / 4900) loss: 1.363185
(Iteration 2231 / 4900) loss: 1.429711
(Iteration 2241 / 4900) loss: 1.260317
(Iteration 2251 / 4900) loss: 1.194974
(Iteration 2261 / 4900) loss: 1.510589
(Iteration 2271 / 4900) loss: 1.448915
(Iteration 2281 / 4900) loss: 1.351496
(Iteration 2291 / 4900) loss: 1.213675
(Iteration 2301 / 4900) loss: 1.429426
(Iteration 2311 / 4900) loss: 1.448498
(Iteration 2321 / 4900) loss: 1.505244
(Iteration 2331 / 4900) loss: 1.188886
(Iteration 2341 / 4900) loss: 1.161387
(Iteration 2351 / 4900) loss: 1.449812
(Iteration 2361 / 4900) loss: 1.216727
(Iteration 2371 / 4900) loss: 1.282557
(Iteration 2381 / 4900) loss: 1.398147
(Iteration 2391 / 4900) loss: 1.522072
(Iteration 2401 / 4900) loss: 1.116810
(Iteration 2411 / 4900) loss: 1.300851
(Iteration 2421 / 4900) loss: 1.282102
(Iteration 2431 / 4900) loss: 1.341910
(Iteration 2441 / 4900) loss: 1.343531
```

```
(Epoch 5 / 10) train acc: 0.520000; val_acc: 0.499000
(Iteration 2451 / 4900) loss: 1.260259
(Iteration 2461 / 4900) loss: 1.344576
(Iteration 2471 / 4900) loss: 1.138951
(Iteration 2481 / 4900) loss: 1.484786
(Iteration 2491 / 4900) loss: 1.198497
(Iteration 2501 / 4900) loss: 1.348365
(Iteration 2511 / 4900) loss: 1.274057
(Iteration 2521 / 4900) loss: 1.119686
(Iteration 2531 / 4900) loss: 1.172003
(Iteration 2541 / 4900) loss: 1.211048
(Iteration 2551 / 4900) loss: 1.372921
(Iteration 2561 / 4900) loss: 1.307111
(Iteration 2571 / 4900) loss: 1.448803
(Iteration 2581 / 4900) loss: 1.320419
(Iteration 2591 / 4900) loss: 1.157638
(Iteration 2601 / 4900) loss: 1.335762
(Iteration 2611 / 4900) loss: 1.383663
(Iteration 2621 / 4900) loss: 1.325651
(Iteration 2631 / 4900) loss: 1.276797
(Iteration 2641 / 4900) loss: 1.175317
(Iteration 2651 / 4900) loss: 1.195821
(Iteration 2661 / 4900) loss: 1.264455
(Iteration 2671 / 4900) loss: 1.289261
(Iteration 2681 / 4900) loss: 1.407857
(Iteration 2691 / 4900) loss: 1.308180
(Iteration 2701 / 4900) loss: 1.383557
(Iteration 2711 / 4900) loss: 1.262342
(Iteration 2721 / 4900) loss: 1.311485
(Iteration 2731 / 4900) loss: 1.289972
(Iteration 2741 / 4900) loss: 1.045919
(Iteration 2751 / 4900) loss: 1.443759
(Iteration 2761 / 4900) loss: 1.402413
(Iteration 2771 / 4900) loss: 1.276887
(Iteration 2781 / 4900) loss: 1.259007
(Iteration 2791 / 4900) loss: 1.176262
(Iteration 2801 / 4900) loss: 1.357269
(Iteration 2811 / 4900) loss: 1.322995
(Iteration 2821 / 4900) loss: 1.485848
(Iteration 2831 / 4900) loss: 1.274852
(Iteration 2841 / 4900) loss: 1.256124
(Iteration 2851 / 4900) loss: 1.156489
(Iteration 2861 / 4900) loss: 1.183315
(Iteration 2871 / 4900) loss: 1.483686
(Iteration 2881 / 4900) loss: 1.309504
(Iteration 2891 / 4900) loss: 1.150435
(Iteration 2901 / 4900) loss: 1.484381
(Iteration 2911 / 4900) loss: 1.279348
```

```
(Iteration 2921 / 4900) loss: 1.125447
(Iteration 2931 / 4900) loss: 1.287655
(Epoch 6 / 10) train acc: 0.544000; val_acc: 0.501000
(Iteration 2941 / 4900) loss: 1.214383
(Iteration 2951 / 4900) loss: 1.317309
(Iteration 2961 / 4900) loss: 1.467561
(Iteration 2971 / 4900) loss: 1.251787
(Iteration 2981 / 4900) loss: 1.371147
(Iteration 2991 / 4900) loss: 1.211347
(Iteration 3001 / 4900) loss: 1.178098
(Iteration 3011 / 4900) loss: 1.463114
(Iteration 3021 / 4900) loss: 1.133168
(Iteration 3031 / 4900) loss: 1.249145
(Iteration 3041 / 4900) loss: 1.186823
(Iteration 3051 / 4900) loss: 1.295622
(Iteration 3061 / 4900) loss: 1.141418
(Iteration 3071 / 4900) loss: 1.212951
(Iteration 3081 / 4900) loss: 1.324655
(Iteration 3091 / 4900) loss: 1.334032
(Iteration 3101 / 4900) loss: 1.408772
(Iteration 3111 / 4900) loss: 1.248382
(Iteration 3121 / 4900) loss: 1.228536
(Iteration 3131 / 4900) loss: 1.311706
(Iteration 3141 / 4900) loss: 1.132722
(Iteration 3151 / 4900) loss: 1.258507
(Iteration 3161 / 4900) loss: 1.339088
(Iteration 3171 / 4900) loss: 1.246927
(Iteration 3181 / 4900) loss: 1.601654
(Iteration 3191 / 4900) loss: 1.184175
(Iteration 3201 / 4900) loss: 1.335045
(Iteration 3211 / 4900) loss: 1.217686
(Iteration 3221 / 4900) loss: 1.373578
(Iteration 3231 / 4900) loss: 1.332960
(Iteration 3241 / 4900) loss: 1.382043
(Iteration 3251 / 4900) loss: 1.143779
(Iteration 3261 / 4900) loss: 1.187749
(Iteration 3271 / 4900) loss: 1.496739
(Iteration 3281 / 4900) loss: 1.263161
(Iteration 3291 / 4900) loss: 1.217443
(Iteration 3301 / 4900) loss: 1.090973
(Iteration 3311 / 4900) loss: 1.265088
(Iteration 3321 / 4900) loss: 1.420308
(Iteration 3331 / 4900) loss: 1.231870
(Iteration 3341 / 4900) loss: 1.359660
(Iteration 3351 / 4900) loss: 1.235078
(Iteration 3361 / 4900) loss: 1.288709
(Iteration 3371 / 4900) loss: 1.178354
(Iteration 3381 / 4900) loss: 1.422973
```

```
(Iteration 3391 / 4900) loss: 1.408243
(Iteration 3401 / 4900) loss: 1.421924
(Iteration 3411 / 4900) loss: 1.369532
(Iteration 3421 / 4900) loss: 1.147469
(Epoch 7 / 10) train acc: 0.554000; val_acc: 0.499000
(Iteration 3431 / 4900) loss: 1.186622
(Iteration 3441 / 4900) loss: 1.330724
(Iteration 3451 / 4900) loss: 1.075731
(Iteration 3461 / 4900) loss: 1.220529
(Iteration 3471 / 4900) loss: 1.423630
(Iteration 3481 / 4900) loss: 1.471127
(Iteration 3491 / 4900) loss: 1.400557
(Iteration 3501 / 4900) loss: 1.313685
(Iteration 3511 / 4900) loss: 1.362723
(Iteration 3521 / 4900) loss: 1.318856
(Iteration 3531 / 4900) loss: 1.266012
(Iteration 3541 / 4900) loss: 1.245153
(Iteration 3551 / 4900) loss: 1.215291
(Iteration 3561 / 4900) loss: 1.209083
(Iteration 3571 / 4900) loss: 1.361239
(Iteration 3581 / 4900) loss: 1.142914
(Iteration 3591 / 4900) loss: 1.202139
(Iteration 3601 / 4900) loss: 1.236793
(Iteration 3611 / 4900) loss: 1.259760
(Iteration 3621 / 4900) loss: 1.086967
(Iteration 3631 / 4900) loss: 1.121066
(Iteration 3641 / 4900) loss: 1.368995
(Iteration 3651 / 4900) loss: 1.125699
(Iteration 3661 / 4900) loss: 1.100391
(Iteration 3671 / 4900) loss: 1.155656
(Iteration 3681 / 4900) loss: 1.002836
(Iteration 3691 / 4900) loss: 1.187961
(Iteration 3701 / 4900) loss: 1.335804
(Iteration 3711 / 4900) loss: 1.045666
(Iteration 3721 / 4900) loss: 1.264709
(Iteration 3731 / 4900) loss: 1.243133
(Iteration 3741 / 4900) loss: 1.013904
(Iteration 3751 / 4900) loss: 1.165042
(Iteration 3761 / 4900) loss: 1.279335
(Iteration 3771 / 4900) loss: 1.452350
(Iteration 3781 / 4900) loss: 1.221436
(Iteration 3791 / 4900) loss: 1.223097
(Iteration 3801 / 4900) loss: 1.202067
(Iteration 3811 / 4900) loss: 1.098291
(Iteration 3821 / 4900) loss: 1.075380
(Iteration 3831 / 4900) loss: 1.262182
(Iteration 3841 / 4900) loss: 1.204827
(Iteration 3851 / 4900) loss: 1.275924
```

```
(Iteration 3861 / 4900) loss: 1.241201
(Iteration 3871 / 4900) loss: 1.117853
(Iteration 3881 / 4900) loss: 1.221101
(Iteration 3891 / 4900) loss: 1.262547
(Iteration 3901 / 4900) loss: 1.219638
(Iteration 3911 / 4900) loss: 1.353291
(Epoch 8 / 10) train acc: 0.608000; val_acc: 0.499000
(Iteration 3921 / 4900) loss: 1.104519
(Iteration 3931 / 4900) loss: 1.146712
(Iteration 3941 / 4900) loss: 1.231864
(Iteration 3951 / 4900) loss: 1.322635
(Iteration 3961 / 4900) loss: 1.259750
(Iteration 3971 / 4900) loss: 0.971409
(Iteration 3981 / 4900) loss: 1.323659
(Iteration 3991 / 4900) loss: 1.049053
(Iteration 4001 / 4900) loss: 1.243922
(Iteration 4011 / 4900) loss: 1.094982
(Iteration 4021 / 4900) loss: 1.398716
(Iteration 4031 / 4900) loss: 1.003354
(Iteration 4041 / 4900) loss: 1.308806
(Iteration 4051 / 4900) loss: 1.237166
(Iteration 4061 / 4900) loss: 1.097408
(Iteration 4071 / 4900) loss: 1.101556
(Iteration 4081 / 4900) loss: 1.056852
(Iteration 4091 / 4900) loss: 1.189157
(Iteration 4101 / 4900) loss: 1.177985
(Iteration 4111 / 4900) loss: 1.155417
(Iteration 4121 / 4900) loss: 0.995625
(Iteration 4131 / 4900) loss: 1.133113
(Iteration 4141 / 4900) loss: 1.253114
(Iteration 4151 / 4900) loss: 1.365883
(Iteration 4161 / 4900) loss: 1.060644
(Iteration 4171 / 4900) loss: 1.200870
(Iteration 4181 / 4900) loss: 1.006577
(Iteration 4191 / 4900) loss: 0.955298
(Iteration 4201 / 4900) loss: 1.209030
(Iteration 4211 / 4900) loss: 1.329195
(Iteration 4221 / 4900) loss: 1.239410
(Iteration 4231 / 4900) loss: 1.336638
(Iteration 4241 / 4900) loss: 1.212623
(Iteration 4251 / 4900) loss: 1.446113
(Iteration 4261 / 4900) loss: 1.193407
(Iteration 4271 / 4900) loss: 1.097583
(Iteration 4281 / 4900) loss: 1.090085
(Iteration 4291 / 4900) loss: 1.154888
(Iteration 4301 / 4900) loss: 1.124682
(Iteration 4311 / 4900) loss: 1.236375
(Iteration 4321 / 4900) loss: 1.372745
```

```
(Iteration 4331 / 4900) loss: 1.095938
(Iteration 4341 / 4900) loss: 1.266866
(Iteration 4351 / 4900) loss: 1.115134
(Iteration 4361 / 4900) loss: 1.118071
(Iteration 4371 / 4900) loss: 1.022754
(Iteration 4381 / 4900) loss: 1.431633
(Iteration 4391 / 4900) loss: 1.270301
(Iteration 4401 / 4900) loss: 1.328931
(Epoch 9 / 10) train acc: 0.543000; val_acc: 0.501000
(Iteration 4411 / 4900) loss: 1.283000
(Iteration 4421 / 4900) loss: 1.328141
(Iteration 4431 / 4900) loss: 1.314314
(Iteration 4441 / 4900) loss: 1.265646
(Iteration 4451 / 4900) loss: 0.996551
(Iteration 4461 / 4900) loss: 1.201751
(Iteration 4471 / 4900) loss: 1.139403
(Iteration 4481 / 4900) loss: 1.152806
(Iteration 4491 / 4900) loss: 1.326931
(Iteration 4501 / 4900) loss: 1.235099
(Iteration 4511 / 4900) loss: 1.268829
(Iteration 4521 / 4900) loss: 1.182240
(Iteration 4531 / 4900) loss: 1.045177
(Iteration 4541 / 4900) loss: 1.136071
(Iteration 4551 / 4900) loss: 1.219877
(Iteration 4561 / 4900) loss: 1.216129
(Iteration 4571 / 4900) loss: 1.075152
(Iteration 4581 / 4900) loss: 1.144516
(Iteration 4591 / 4900) loss: 1.228011
(Iteration 4601 / 4900) loss: 1.078181
(Iteration 4611 / 4900) loss: 1.191456
(Iteration 4621 / 4900) loss: 1.420029
(Iteration 4631 / 4900) loss: 0.979150
(Iteration 4641 / 4900) loss: 1.005221
(Iteration 4651 / 4900) loss: 1.113908
(Iteration 4661 / 4900) loss: 1.147429
(Iteration 4671 / 4900) loss: 1.197200
(Iteration 4681 / 4900) loss: 0.950847
(Iteration 4691 / 4900) loss: 1.017578
(Iteration 4701 / 4900) loss: 1.213653
(Iteration 4711 / 4900) loss: 1.139174
(Iteration 4721 / 4900) loss: 0.913849
(Iteration 4731 / 4900) loss: 1.204906
(Iteration 4741 / 4900) loss: 1.107700
(Iteration 4751 / 4900) loss: 1.248307
(Iteration 4761 / 4900) loss: 1.202188
(Iteration 4771 / 4900) loss: 0.989315
(Iteration 4781 / 4900) loss: 1.179201
(Iteration 4791 / 4900) loss: 1.036194
```

```
(Iteration 4801 / 4900) loss: 1.247814
(Iteration 4811 / 4900) loss: 1.444238
(Iteration 4821 / 4900) loss: 1.017144
(Iteration 4831 / 4900) loss: 1.047292
(Iteration 4841 / 4900) loss: 1.205764
(Iteration 4851 / 4900) loss: 0.995429
(Iteration 4861 / 4900) loss: 1.265748
(Iteration 4871 / 4900) loss: 0.926069
(Iteration 4881 / 4900) loss: 1.034549
(Iteration 4891 / 4900) loss: 1.189110
(Epoch 10 / 10) train acc: 0.584000; val_acc: 0.509000
(Iteration 1 / 4900) loss: 2.302579
(Epoch 0 / 10) train acc: 0.098000; val_acc: 0.082000
(Iteration 11 / 4900) loss: 2.263288
(Iteration 21 / 4900) loss: 2.138136
(Iteration 31 / 4900) loss: 2.171478
(Iteration 41 / 4900) loss: 1.953885
(Iteration 51 / 4900) loss: 1.989146
(Iteration 61 / 4900) loss: 2.149833
(Iteration 71 / 4900) loss: 1.979445
(Iteration 81 / 4900) loss: 2.066438
(Iteration 91 / 4900) loss: 1.987461
(Iteration 101 / 4900) loss: 1.851624
(Iteration 111 / 4900) loss: 2.031135
(Iteration 121 / 4900) loss: 2.117612
(Iteration 131 / 4900) loss: 1.979218
(Iteration 141 / 4900) loss: 1.952428
(Iteration 151 / 4900) loss: 1.876301
(Iteration 161 / 4900) loss: 1.855230
(Iteration 171 / 4900) loss: 1.831227
(Iteration 181 / 4900) loss: 1.815858
(Iteration 191 / 4900) loss: 1.863273
(Iteration 201 / 4900) loss: 1.807651
(Iteration 211 / 4900) loss: 1.830891
(Iteration 221 / 4900) loss: 1.896227
(Iteration 231 / 4900) loss: 1.886835
(Iteration 241 / 4900) loss: 1.791483
(Iteration 251 / 4900) loss: 1.752026
(Iteration 261 / 4900) loss: 1.740831
(Iteration 271 / 4900) loss: 1.914379
(Iteration 281 / 4900) loss: 1.809495
(Iteration 291 / 4900) loss: 1.768777
(Iteration 301 / 4900) loss: 1.822264
(Iteration 311 / 4900) loss: 1.748350
(Iteration 321 / 4900) loss: 1.851872
(Iteration 331 / 4900) loss: 1.639451
(Iteration 341 / 4900) loss: 1.964264
(Iteration 351 / 4900) loss: 1.703544
```

```
(Iteration 361 / 4900) loss: 1.820520
(Iteration 371 / 4900) loss: 1.686727
(Iteration 381 / 4900) loss: 1.758716
(Iteration 391 / 4900) loss: 1.658564
(Iteration 401 / 4900) loss: 1.856476
(Iteration 411 / 4900) loss: 1.712600
(Iteration 421 / 4900) loss: 1.861229
(Iteration 431 / 4900) loss: 1.705454
(Iteration 441 / 4900) loss: 1.710605
(Iteration 451 / 4900) loss: 1.791182
(Iteration 461 / 4900) loss: 1.605503
(Iteration 471 / 4900) loss: 1.825153
(Iteration 481 / 4900) loss: 1.578681
(Epoch 1 / 10) train acc: 0.380000; val_acc: 0.382000
(Iteration 491 / 4900) loss: 1.953642
(Iteration 501 / 4900) loss: 1.795932
(Iteration 511 / 4900) loss: 1.724960
(Iteration 521 / 4900) loss: 1.699082
(Iteration 531 / 4900) loss: 1.524207
(Iteration 541 / 4900) loss: 1.801912
(Iteration 551 / 4900) loss: 1.593194
(Iteration 561 / 4900) loss: 1.709955
(Iteration 571 / 4900) loss: 1.558437
(Iteration 581 / 4900) loss: 1.700355
(Iteration 591 / 4900) loss: 1.788802
(Iteration 601 / 4900) loss: 1.557867
(Iteration 611 / 4900) loss: 1.650379
(Iteration 621 / 4900) loss: 1.644969
(Iteration 631 / 4900) loss: 1.573558
(Iteration 641 / 4900) loss: 1.382075
(Iteration 651 / 4900) loss: 1.669433
(Iteration 661 / 4900) loss: 1.465719
(Iteration 671 / 4900) loss: 1.477593
(Iteration 681 / 4900) loss: 1.581130
(Iteration 691 / 4900) loss: 1.447223
(Iteration 701 / 4900) loss: 1.813811
(Iteration 711 / 4900) loss: 1.594998
(Iteration 721 / 4900) loss: 1.516417
(Iteration 731 / 4900) loss: 1.799666
(Iteration 741 / 4900) loss: 1.477733
(Iteration 751 / 4900) loss: 1.518844
(Iteration 761 / 4900) loss: 1.507906
(Iteration 771 / 4900) loss: 1.409687
(Iteration 781 / 4900) loss: 1.628360
(Iteration 791 / 4900) loss: 1.577290
(Iteration 801 / 4900) loss: 1.465984
(Iteration 811 / 4900) loss: 1.415487
(Iteration 821 / 4900) loss: 1.546464
```

```
(Iteration 831 / 4900) loss: 1.498845
(Iteration 841 / 4900) loss: 1.655292
(Iteration 851 / 4900) loss: 1.652279
(Iteration 861 / 4900) loss: 1.458411
(Iteration 871 / 4900) loss: 1.454409
(Iteration 881 / 4900) loss: 1.693173
(Iteration 891 / 4900) loss: 1.522189
(Iteration 901 / 4900) loss: 1.507875
(Iteration 911 / 4900) loss: 1.632543
(Iteration 921 / 4900) loss: 1.415022
(Iteration 931 / 4900) loss: 1.381976
(Iteration 941 / 4900) loss: 1.482584
(Iteration 951 / 4900) loss: 1.764796
(Iteration 961 / 4900) loss: 1.520823
(Iteration 971 / 4900) loss: 1.526274
(Epoch 2 / 10) train acc: 0.454000; val_acc: 0.443000
(Iteration 981 / 4900) loss: 1.480997
(Iteration 991 / 4900) loss: 1.402174
(Iteration 1001 / 4900) loss: 1.622387
(Iteration 1011 / 4900) loss: 1.468758
(Iteration 1021 / 4900) loss: 1.512833
(Iteration 1031 / 4900) loss: 1.585444
(Iteration 1041 / 4900) loss: 1.899467
(Iteration 1051 / 4900) loss: 1.466480
(Iteration 1061 / 4900) loss: 1.398732
(Iteration 1071 / 4900) loss: 1.510089
(Iteration 1081 / 4900) loss: 1.500320
(Iteration 1091 / 4900) loss: 1.438742
(Iteration 1101 / 4900) loss: 1.379308
(Iteration 1111 / 4900) loss: 1.433753
(Iteration 1121 / 4900) loss: 1.514375
(Iteration 1131 / 4900) loss: 1.620416
(Iteration 1141 / 4900) loss: 1.627689
(Iteration 1151 / 4900) loss: 1.630426
(Iteration 1161 / 4900) loss: 1.463490
(Iteration 1171 / 4900) loss: 1.390222
(Iteration 1181 / 4900) loss: 1.508381
(Iteration 1191 / 4900) loss: 1.486346
(Iteration 1201 / 4900) loss: 1.518007
(Iteration 1211 / 4900) loss: 1.478609
(Iteration 1221 / 4900) loss: 1.418226
(Iteration 1231 / 4900) loss: 1.246083
(Iteration 1241 / 4900) loss: 1.433433
(Iteration 1251 / 4900) loss: 1.530595
(Iteration 1261 / 4900) loss: 1.459447
(Iteration 1271 / 4900) loss: 1.442537
(Iteration 1281 / 4900) loss: 1.409070
(Iteration 1291 / 4900) loss: 1.425982
```

```
(Iteration 1301 / 4900) loss: 1.296923
(Iteration 1311 / 4900) loss: 1.528801
(Iteration 1321 / 4900) loss: 1.326270
(Iteration 1331 / 4900) loss: 1.595823
(Iteration 1341 / 4900) loss: 1.423042
(Iteration 1351 / 4900) loss: 1.419486
(Iteration 1361 / 4900) loss: 1.421912
(Iteration 1371 / 4900) loss: 1.529718
(Iteration 1381 / 4900) loss: 1.387967
(Iteration 1391 / 4900) loss: 1.506174
(Iteration 1401 / 4900) loss: 1.496836
(Iteration 1411 / 4900) loss: 1.460742
(Iteration 1421 / 4900) loss: 1.451095
(Iteration 1431 / 4900) loss: 1.465427
(Iteration 1441 / 4900) loss: 1.481713
(Iteration 1451 / 4900) loss: 1.564131
(Iteration 1461 / 4900) loss: 1.448242
(Epoch 3 / 10) train acc: 0.491000; val_acc: 0.456000
(Iteration 1471 / 4900) loss: 1.309265
(Iteration 1481 / 4900) loss: 1.258623
(Iteration 1491 / 4900) loss: 1.274327
(Iteration 1501 / 4900) loss: 1.357446
(Iteration 1511 / 4900) loss: 1.376531
(Iteration 1521 / 4900) loss: 1.300976
(Iteration 1531 / 4900) loss: 1.521028
(Iteration 1541 / 4900) loss: 1.457336
(Iteration 1551 / 4900) loss: 1.493736
(Iteration 1561 / 4900) loss: 1.478591
(Iteration 1571 / 4900) loss: 1.344923
(Iteration 1581 / 4900) loss: 1.305575
(Iteration 1591 / 4900) loss: 1.502908
(Iteration 1601 / 4900) loss: 1.312395
(Iteration 1611 / 4900) loss: 1.299913
(Iteration 1621 / 4900) loss: 1.239254
(Iteration 1631 / 4900) loss: 1.261032
(Iteration 1641 / 4900) loss: 1.426280
(Iteration 1651 / 4900) loss: 1.531628
(Iteration 1661 / 4900) loss: 1.147879
(Iteration 1671 / 4900) loss: 1.279885
(Iteration 1681 / 4900) loss: 1.321464
(Iteration 1691 / 4900) loss: 1.415200
(Iteration 1701 / 4900) loss: 1.355446
(Iteration 1711 / 4900) loss: 1.399471
(Iteration 1721 / 4900) loss: 1.439430
(Iteration 1731 / 4900) loss: 1.448969
(Iteration 1741 / 4900) loss: 1.487500
(Iteration 1751 / 4900) loss: 1.332788
(Iteration 1761 / 4900) loss: 1.348042
```

```
(Iteration 1771 / 4900) loss: 1.319656
(Iteration 1781 / 4900) loss: 1.586162
(Iteration 1791 / 4900) loss: 1.456838
(Iteration 1801 / 4900) loss: 1.386685
(Iteration 1811 / 4900) loss: 1.329236
(Iteration 1821 / 4900) loss: 1.397903
(Iteration 1831 / 4900) loss: 1.331132
(Iteration 1841 / 4900) loss: 1.508440
(Iteration 1851 / 4900) loss: 1.318521
(Iteration 1861 / 4900) loss: 1.249013
(Iteration 1871 / 4900) loss: 1.327673
(Iteration 1881 / 4900) loss: 1.339225
(Iteration 1891 / 4900) loss: 1.348913
(Iteration 1901 / 4900) loss: 1.423995
(Iteration 1911 / 4900) loss: 1.485054
(Iteration 1921 / 4900) loss: 1.735674
(Iteration 1931 / 4900) loss: 1.421469
(Iteration 1941 / 4900) loss: 1.384430
(Iteration 1951 / 4900) loss: 1.375421
(Epoch 4 / 10) train acc: 0.499000; val_acc: 0.495000
(Iteration 1961 / 4900) loss: 1.472324
(Iteration 1971 / 4900) loss: 1.285235
(Iteration 1981 / 4900) loss: 1.491531
(Iteration 1991 / 4900) loss: 1.472936
(Iteration 2001 / 4900) loss: 1.332092
(Iteration 2011 / 4900) loss: 1.312288
(Iteration 2021 / 4900) loss: 1.228336
(Iteration 2031 / 4900) loss: 1.337464
(Iteration 2041 / 4900) loss: 1.251920
(Iteration 2051 / 4900) loss: 1.450257
(Iteration 2061 / 4900) loss: 1.408015
(Iteration 2071 / 4900) loss: 1.334451
(Iteration 2081 / 4900) loss: 1.295071
(Iteration 2091 / 4900) loss: 1.176971
(Iteration 2101 / 4900) loss: 1.495519
(Iteration 2111 / 4900) loss: 1.323836
(Iteration 2121 / 4900) loss: 1.443263
(Iteration 2131 / 4900) loss: 1.211567
(Iteration 2141 / 4900) loss: 1.296886
(Iteration 2151 / 4900) loss: 1.303668
(Iteration 2161 / 4900) loss: 1.229744
(Iteration 2171 / 4900) loss: 1.294973
(Iteration 2181 / 4900) loss: 1.248371
(Iteration 2191 / 4900) loss: 1.327530
(Iteration 2201 / 4900) loss: 1.294739
(Iteration 2211 / 4900) loss: 1.294039
(Iteration 2221 / 4900) loss: 1.402815
(Iteration 2231 / 4900) loss: 1.184410
```

```
(Iteration 2241 / 4900) loss: 1.497684
(Iteration 2251 / 4900) loss: 1.172548
(Iteration 2261 / 4900) loss: 1.320736
(Iteration 2271 / 4900) loss: 1.346517
(Iteration 2281 / 4900) loss: 1.374715
(Iteration 2291 / 4900) loss: 1.079734
(Iteration 2301 / 4900) loss: 1.385098
(Iteration 2311 / 4900) loss: 1.405681
(Iteration 2321 / 4900) loss: 1.260395
(Iteration 2331 / 4900) loss: 1.461714
(Iteration 2341 / 4900) loss: 1.107380
(Iteration 2351 / 4900) loss: 1.341142
(Iteration 2361 / 4900) loss: 1.213789
(Iteration 2371 / 4900) loss: 1.185108
(Iteration 2381 / 4900) loss: 1.321432
(Iteration 2391 / 4900) loss: 1.293397
(Iteration 2401 / 4900) loss: 1.345236
(Iteration 2411 / 4900) loss: 1.449389
(Iteration 2421 / 4900) loss: 1.493095
(Iteration 2431 / 4900) loss: 1.412031
(Iteration 2441 / 4900) loss: 1.382191
(Epoch 5 / 10) train acc: 0.540000; val_acc: 0.493000
(Iteration 2451 / 4900) loss: 1.570308
(Iteration 2461 / 4900) loss: 1.215462
(Iteration 2471 / 4900) loss: 1.256871
(Iteration 2481 / 4900) loss: 1.346712
(Iteration 2491 / 4900) loss: 1.272235
(Iteration 2501 / 4900) loss: 1.177628
(Iteration 2511 / 4900) loss: 1.291934
(Iteration 2521 / 4900) loss: 1.165191
(Iteration 2531 / 4900) loss: 1.062556
(Iteration 2541 / 4900) loss: 1.096123
(Iteration 2551 / 4900) loss: 1.366646
(Iteration 2561 / 4900) loss: 1.204962
(Iteration 2571 / 4900) loss: 1.389663
(Iteration 2581 / 4900) loss: 1.296828
(Iteration 2591 / 4900) loss: 1.146120
(Iteration 2601 / 4900) loss: 1.267434
(Iteration 2611 / 4900) loss: 1.446972
(Iteration 2621 / 4900) loss: 1.170389
(Iteration 2631 / 4900) loss: 1.097503
(Iteration 2641 / 4900) loss: 1.365209
(Iteration 2651 / 4900) loss: 1.118939
(Iteration 2661 / 4900) loss: 1.322570
(Iteration 2671 / 4900) loss: 1.410033
(Iteration 2681 / 4900) loss: 1.291036
(Iteration 2691 / 4900) loss: 1.263200
(Iteration 2701 / 4900) loss: 1.370256
```

```
(Iteration 2711 / 4900) loss: 1.153555
(Iteration 2721 / 4900) loss: 1.529045
(Iteration 2731 / 4900) loss: 1.150065
(Iteration 2741 / 4900) loss: 1.217979
(Iteration 2751 / 4900) loss: 1.481735
(Iteration 2761 / 4900) loss: 1.269495
(Iteration 2771 / 4900) loss: 1.197621
(Iteration 2781 / 4900) loss: 1.197283
(Iteration 2791 / 4900) loss: 1.460042
(Iteration 2801 / 4900) loss: 1.220072
(Iteration 2811 / 4900) loss: 0.979504
(Iteration 2821 / 4900) loss: 1.312274
(Iteration 2831 / 4900) loss: 1.113594
(Iteration 2841 / 4900) loss: 1.581997
(Iteration 2851 / 4900) loss: 1.223299
(Iteration 2861 / 4900) loss: 1.118809
(Iteration 2871 / 4900) loss: 1.259151
(Iteration 2881 / 4900) loss: 1.195841
(Iteration 2891 / 4900) loss: 1.402254
(Iteration 2901 / 4900) loss: 1.226014
(Iteration 2911 / 4900) loss: 1.268687
(Iteration 2921 / 4900) loss: 1.359323
(Iteration 2931 / 4900) loss: 1.200541
(Epoch 6 / 10) train acc: 0.547000; val_acc: 0.518000
(Iteration 2941 / 4900) loss: 1.260833
(Iteration 2951 / 4900) loss: 1.208231
(Iteration 2961 / 4900) loss: 1.289187
(Iteration 2971 / 4900) loss: 1.170532
(Iteration 2981 / 4900) loss: 1.125934
(Iteration 2991 / 4900) loss: 1.240492
(Iteration 3001 / 4900) loss: 1.168284
(Iteration 3011 / 4900) loss: 1.071525
(Iteration 3021 / 4900) loss: 1.275950
(Iteration 3031 / 4900) loss: 1.433439
(Iteration 3041 / 4900) loss: 1.350444
(Iteration 3051 / 4900) loss: 1.269217
(Iteration 3061 / 4900) loss: 1.233632
(Iteration 3071 / 4900) loss: 1.325258
(Iteration 3081 / 4900) loss: 1.483712
(Iteration 3091 / 4900) loss: 1.083198
(Iteration 3101 / 4900) loss: 1.135764
(Iteration 3111 / 4900) loss: 1.258016
(Iteration 3121 / 4900) loss: 1.319393
(Iteration 3131 / 4900) loss: 1.228652
(Iteration 3141 / 4900) loss: 1.229316
(Iteration 3151 / 4900) loss: 1.362725
(Iteration 3161 / 4900) loss: 1.054062
(Iteration 3171 / 4900) loss: 1.294382
```

```
(Iteration 3181 / 4900) loss: 1.231719
(Iteration 3191 / 4900) loss: 1.186794
(Iteration 3201 / 4900) loss: 1.312522
(Iteration 3211 / 4900) loss: 1.300194
(Iteration 3221 / 4900) loss: 1.249956
(Iteration 3231 / 4900) loss: 1.432410
(Iteration 3241 / 4900) loss: 1.361863
(Iteration 3251 / 4900) loss: 1.099626
(Iteration 3261 / 4900) loss: 1.194941
(Iteration 3271 / 4900) loss: 1.222233
(Iteration 3281 / 4900) loss: 1.266353
(Iteration 3291 / 4900) loss: 1.344885
(Iteration 3301 / 4900) loss: 1.016110
(Iteration 3311 / 4900) loss: 1.177339
(Iteration 3321 / 4900) loss: 1.342365
(Iteration 3331 / 4900) loss: 0.953378
(Iteration 3341 / 4900) loss: 1.224045
(Iteration 3351 / 4900) loss: 1.101662
(Iteration 3361 / 4900) loss: 1.109399
(Iteration 3371 / 4900) loss: 1.139742
(Iteration 3381 / 4900) loss: 1.375032
(Iteration 3391 / 4900) loss: 1.145790
(Iteration 3401 / 4900) loss: 1.392739
(Iteration 3411 / 4900) loss: 1.083741
(Iteration 3421 / 4900) loss: 1.086867
(Epoch 7 / 10) train acc: 0.560000; val_acc: 0.510000
(Iteration 3431 / 4900) loss: 1.196630
(Iteration 3441 / 4900) loss: 1.386061
(Iteration 3451 / 4900) loss: 1.404534
(Iteration 3461 / 4900) loss: 1.185131
(Iteration 3471 / 4900) loss: 1.115379
(Iteration 3481 / 4900) loss: 1.247552
(Iteration 3491 / 4900) loss: 1.511843
(Iteration 3501 / 4900) loss: 1.179134
(Iteration 3511 / 4900) loss: 1.052103
(Iteration 3521 / 4900) loss: 1.149344
(Iteration 3531 / 4900) loss: 1.095501
(Iteration 3541 / 4900) loss: 1.235335
(Iteration 3551 / 4900) loss: 1.036581
(Iteration 3561 / 4900) loss: 1.244451
(Iteration 3571 / 4900) loss: 1.269194
(Iteration 3581 / 4900) loss: 1.088162
(Iteration 3591 / 4900) loss: 1.247045
(Iteration 3601 / 4900) loss: 1.186931
(Iteration 3611 / 4900) loss: 1.258436
(Iteration 3621 / 4900) loss: 1.094879
(Iteration 3631 / 4900) loss: 1.219553
(Iteration 3641 / 4900) loss: 1.241944
```

```
(Iteration 3651 / 4900) loss: 1.235354
(Iteration 3661 / 4900) loss: 1.201805
(Iteration 3671 / 4900) loss: 1.295830
(Iteration 3681 / 4900) loss: 1.191223
(Iteration 3691 / 4900) loss: 1.223157
(Iteration 3701 / 4900) loss: 0.962805
(Iteration 3711 / 4900) loss: 1.218421
(Iteration 3721 / 4900) loss: 1.061552
(Iteration 3731 / 4900) loss: 1.081655
(Iteration 3741 / 4900) loss: 1.277698
(Iteration 3751 / 4900) loss: 1.230024
(Iteration 3761 / 4900) loss: 1.295481
(Iteration 3771 / 4900) loss: 1.198295
(Iteration 3781 / 4900) loss: 0.898864
(Iteration 3791 / 4900) loss: 1.089338
(Iteration 3801 / 4900) loss: 1.223139
(Iteration 3811 / 4900) loss: 1.052246
(Iteration 3821 / 4900) loss: 1.260086
(Iteration 3831 / 4900) loss: 1.283309
(Iteration 3841 / 4900) loss: 1.055590
(Iteration 3851 / 4900) loss: 1.161460
(Iteration 3861 / 4900) loss: 1.205579
(Iteration 3871 / 4900) loss: 1.143515
(Iteration 3881 / 4900) loss: 0.987751
(Iteration 3891 / 4900) loss: 1.350438
(Iteration 3901 / 4900) loss: 1.149015
(Iteration 3911 / 4900) loss: 1.127031
(Epoch 8 / 10) train acc: 0.559000; val_acc: 0.508000
(Iteration 3921 / 4900) loss: 1.170346
(Iteration 3931 / 4900) loss: 1.202209
(Iteration 3941 / 4900) loss: 0.989429
(Iteration 3951 / 4900) loss: 1.058384
(Iteration 3961 / 4900) loss: 1.316061
(Iteration 3971 / 4900) loss: 1.208103
(Iteration 3981 / 4900) loss: 1.223530
(Iteration 3991 / 4900) loss: 1.184795
(Iteration 4001 / 4900) loss: 0.931503
(Iteration 4011 / 4900) loss: 1.231925
(Iteration 4021 / 4900) loss: 1.278803
(Iteration 4031 / 4900) loss: 1.105320
(Iteration 4041 / 4900) loss: 1.110367
(Iteration 4051 / 4900) loss: 1.287533
(Iteration 4061 / 4900) loss: 1.120034
(Iteration 4071 / 4900) loss: 1.028803
(Iteration 4081 / 4900) loss: 1.062100
(Iteration 4091 / 4900) loss: 1.333488
(Iteration 4101 / 4900) loss: 0.986028
(Iteration 4111 / 4900) loss: 1.022374
```

```
(Iteration 4121 / 4900) loss: 1.289872
(Iteration 4131 / 4900) loss: 0.995740
(Iteration 4141 / 4900) loss: 1.256255
(Iteration 4151 / 4900) loss: 1.315806
(Iteration 4161 / 4900) loss: 1.117902
(Iteration 4171 / 4900) loss: 1.365356
(Iteration 4181 / 4900) loss: 1.153467
(Iteration 4191 / 4900) loss: 1.004419
(Iteration 4201 / 4900) loss: 1.103077
(Iteration 4211 / 4900) loss: 1.155612
(Iteration 4221 / 4900) loss: 1.012503
(Iteration 4231 / 4900) loss: 0.907824
(Iteration 4241 / 4900) loss: 1.018084
(Iteration 4251 / 4900) loss: 1.105883
(Iteration 4261 / 4900) loss: 1.078337
(Iteration 4271 / 4900) loss: 1.297055
(Iteration 4281 / 4900) loss: 0.987924
(Iteration 4291 / 4900) loss: 1.080845
(Iteration 4301 / 4900) loss: 1.155329
(Iteration 4311 / 4900) loss: 1.025060
(Iteration 4321 / 4900) loss: 0.996083
(Iteration 4331 / 4900) loss: 1.199529
(Iteration 4341 / 4900) loss: 1.137789
(Iteration 4351 / 4900) loss: 1.150262
(Iteration 4361 / 4900) loss: 1.100271
(Iteration 4371 / 4900) loss: 1.432020
(Iteration 4381 / 4900) loss: 0.972375
(Iteration 4391 / 4900) loss: 1.088745
(Iteration 4401 / 4900) loss: 1.078001
(Epoch 9 / 10) train acc: 0.618000; val_acc: 0.512000
(Iteration 4411 / 4900) loss: 1.176676
(Iteration 4421 / 4900) loss: 1.049461
(Iteration 4431 / 4900) loss: 1.017751
(Iteration 4441 / 4900) loss: 1.185510
(Iteration 4451 / 4900) loss: 1.180599
(Iteration 4461 / 4900) loss: 1.039149
(Iteration 4471 / 4900) loss: 1.077335
(Iteration 4481 / 4900) loss: 1.136653
(Iteration 4491 / 4900) loss: 1.096964
(Iteration 4501 / 4900) loss: 1.116230
(Iteration 4511 / 4900) loss: 1.240248
(Iteration 4521 / 4900) loss: 1.219510
(Iteration 4531 / 4900) loss: 1.106853
(Iteration 4541 / 4900) loss: 1.132785
(Iteration 4551 / 4900) loss: 1.047329
(Iteration 4561 / 4900) loss: 1.151002
(Iteration 4571 / 4900) loss: 0.954886
(Iteration 4581 / 4900) loss: 1.093021
```

```
(Iteration 4591 / 4900) loss: 1.125409
(Iteration 4601 / 4900) loss: 1.108972
(Iteration 4611 / 4900) loss: 0.974342
(Iteration 4621 / 4900) loss: 1.231882
(Iteration 4631 / 4900) loss: 1.143630
(Iteration 4641 / 4900) loss: 1.202078
(Iteration 4651 / 4900) loss: 1.034118
(Iteration 4661 / 4900) loss: 1.071067
(Iteration 4671 / 4900) loss: 0.977918
(Iteration 4681 / 4900) loss: 1.164222
(Iteration 4691 / 4900) loss: 1.060349
(Iteration 4701 / 4900) loss: 0.927975
(Iteration 4711 / 4900) loss: 1.285424
(Iteration 4721 / 4900) loss: 1.024308
(Iteration 4731 / 4900) loss: 1.261796
(Iteration 4741 / 4900) loss: 1.132473
(Iteration 4751 / 4900) loss: 1.184071
(Iteration 4761 / 4900) loss: 1.109269
(Iteration 4771 / 4900) loss: 1.199711
(Iteration 4781 / 4900) loss: 1.157221
(Iteration 4791 / 4900) loss: 1.315727
(Iteration 4801 / 4900) loss: 0.886876
(Iteration 4811 / 4900) loss: 1.079482
(Iteration 4821 / 4900) loss: 1.123232
(Iteration 4831 / 4900) loss: 0.968755
(Iteration 4841 / 4900) loss: 0.908842
(Iteration 4851 / 4900) loss: 1.016140
(Iteration 4861 / 4900) loss: 1.162971
(Iteration 4871 / 4900) loss: 1.227238
(Iteration 4881 / 4900) loss: 0.930354
(Iteration 4891 / 4900) loss: 1.156392
(Epoch 10 / 10) train acc: 0.606000; val_acc: 0.498000
(Iteration 1 / 4900) loss: 2.546257
(Epoch 0 / 10) train acc: 0.115000; val_acc: 0.127000
(Iteration 11 / 4900) loss: 2.082013
(Iteration 21 / 4900) loss: 2.132863
(Iteration 31 / 4900) loss: 1.947835
(Iteration 41 / 4900) loss: 1.911512
(Iteration 51 / 4900) loss: 1.753911
(Iteration 61 / 4900) loss: 1.936809
(Iteration 71 / 4900) loss: 1.910752
(Iteration 81 / 4900) loss: 1.984285
(Iteration 91 / 4900) loss: 1.758001
(Iteration 101 / 4900) loss: 1.765000
(Iteration 111 / 4900) loss: 1.692694
(Iteration 121 / 4900) loss: 1.695091
(Iteration 131 / 4900) loss: 1.817905
(Iteration 141 / 4900) loss: 1.717097
```

```
(Iteration 151 / 4900) loss: 1.671048
(Iteration 161 / 4900) loss: 1.680164
(Iteration 171 / 4900) loss: 1.713430
(Iteration 181 / 4900) loss: 1.646628
(Iteration 191 / 4900) loss: 1.755541
(Iteration 201 / 4900) loss: 1.652631
(Iteration 211 / 4900) loss: 1.850245
(Iteration 221 / 4900) loss: 1.785604
(Iteration 231 / 4900) loss: 1.808160
(Iteration 241 / 4900) loss: 1.700819
(Iteration 251 / 4900) loss: 1.650019
(Iteration 261 / 4900) loss: 1.635646
(Iteration 271 / 4900) loss: 1.797523
(Iteration 281 / 4900) loss: 1.550897
(Iteration 291 / 4900) loss: 1.600501
(Iteration 301 / 4900) loss: 1.751067
(Iteration 311 / 4900) loss: 1.683810
(Iteration 321 / 4900) loss: 1.507774
(Iteration 331 / 4900) loss: 1.542699
(Iteration 341 / 4900) loss: 1.565753
(Iteration 351 / 4900) loss: 1.584888
(Iteration 361 / 4900) loss: 1.472569
(Iteration 371 / 4900) loss: 1.709925
(Iteration 381 / 4900) loss: 1.621521
(Iteration 391 / 4900) loss: 1.454002
(Iteration 401 / 4900) loss: 1.579400
(Iteration 411 / 4900) loss: 1.395121
(Iteration 421 / 4900) loss: 1.761410
(Iteration 431 / 4900) loss: 1.731295
(Iteration 441 / 4900) loss: 1.555646
(Iteration 451 / 4900) loss: 1.428600
(Iteration 461 / 4900) loss: 1.406145
(Iteration 471 / 4900) loss: 1.631226
(Iteration 481 / 4900) loss: 1.617388
(Epoch 1 / 10) train acc: 0.428000; val_acc: 0.440000
(Iteration 491 / 4900) loss: 1.853172
(Iteration 501 / 4900) loss: 1.638494
(Iteration 511 / 4900) loss: 1.486804
(Iteration 521 / 4900) loss: 1.357970
(Iteration 531 / 4900) loss: 1.481791
(Iteration 541 / 4900) loss: 1.450316
(Iteration 551 / 4900) loss: 1.589691
(Iteration 561 / 4900) loss: 1.513210
(Iteration 571 / 4900) loss: 1.543077
(Iteration 581 / 4900) loss: 1.595196
(Iteration 591 / 4900) loss: 1.585395
(Iteration 601 / 4900) loss: 1.611798
(Iteration 611 / 4900) loss: 1.607505
```

```
(Iteration 621 / 4900) loss: 1.439658
(Iteration 631 / 4900) loss: 1.610722
(Iteration 641 / 4900) loss: 1.356410
(Iteration 651 / 4900) loss: 1.400154
(Iteration 661 / 4900) loss: 1.528114
(Iteration 671 / 4900) loss: 1.495214
(Iteration 681 / 4900) loss: 1.371522
(Iteration 691 / 4900) loss: 1.423658
(Iteration 701 / 4900) loss: 1.610648
(Iteration 711 / 4900) loss: 1.667130
(Iteration 721 / 4900) loss: 1.355251
(Iteration 731 / 4900) loss: 1.455326
(Iteration 741 / 4900) loss: 1.527302
(Iteration 751 / 4900) loss: 1.598768
(Iteration 761 / 4900) loss: 1.338384
(Iteration 771 / 4900) loss: 1.658778
(Iteration 781 / 4900) loss: 1.534333
(Iteration 791 / 4900) loss: 1.596718
(Iteration 801 / 4900) loss: 1.556262
(Iteration 811 / 4900) loss: 1.489355
(Iteration 821 / 4900) loss: 1.385469
(Iteration 831 / 4900) loss: 1.563793
(Iteration 841 / 4900) loss: 1.444986
(Iteration 851 / 4900) loss: 1.397864
(Iteration 861 / 4900) loss: 1.418196
(Iteration 871 / 4900) loss: 1.461388
(Iteration 881 / 4900) loss: 1.474191
(Iteration 891 / 4900) loss: 1.366064
(Iteration 901 / 4900) loss: 1.225659
(Iteration 911 / 4900) loss: 1.544018
(Iteration 921 / 4900) loss: 1.580754
(Iteration 931 / 4900) loss: 1.592661
(Iteration 941 / 4900) loss: 1.310029
(Iteration 951 / 4900) loss: 1.434504
(Iteration 961 / 4900) loss: 1.541964
(Iteration 971 / 4900) loss: 1.503303
(Epoch 2 / 10) train acc: 0.497000; val_acc: 0.459000
(Iteration 981 / 4900) loss: 1.398515
(Iteration 991 / 4900) loss: 1.263493
(Iteration 1001 / 4900) loss: 1.484302
(Iteration 1011 / 4900) loss: 1.261774
(Iteration 1021 / 4900) loss: 1.352683
(Iteration 1031 / 4900) loss: 1.446033
(Iteration 1041 / 4900) loss: 1.460219
(Iteration 1051 / 4900) loss: 1.476013
(Iteration 1061 / 4900) loss: 1.378571
(Iteration 1071 / 4900) loss: 1.339363
(Iteration 1081 / 4900) loss: 1.554711
```

```
(Iteration 1091 / 4900) loss: 1.300171
(Iteration 1101 / 4900) loss: 1.505948
(Iteration 1111 / 4900) loss: 1.298996
(Iteration 1121 / 4900) loss: 1.327636
(Iteration 1131 / 4900) loss: 1.448375
(Iteration 1141 / 4900) loss: 1.810500
(Iteration 1151 / 4900) loss: 1.420301
(Iteration 1161 / 4900) loss: 1.304921
(Iteration 1171 / 4900) loss: 1.187828
(Iteration 1181 / 4900) loss: 1.489686
(Iteration 1191 / 4900) loss: 1.363903
(Iteration 1201 / 4900) loss: 1.615860
(Iteration 1211 / 4900) loss: 1.281460
(Iteration 1221 / 4900) loss: 1.535144
(Iteration 1231 / 4900) loss: 1.609861
(Iteration 1241 / 4900) loss: 1.403485
(Iteration 1251 / 4900) loss: 1.408040
(Iteration 1261 / 4900) loss: 1.278563
(Iteration 1271 / 4900) loss: 1.265866
(Iteration 1281 / 4900) loss: 1.284667
(Iteration 1291 / 4900) loss: 1.418082
(Iteration 1301 / 4900) loss: 1.373969
(Iteration 1311 / 4900) loss: 1.555209
(Iteration 1321 / 4900) loss: 1.385065
(Iteration 1331 / 4900) loss: 1.622629
(Iteration 1341 / 4900) loss: 1.253409
(Iteration 1351 / 4900) loss: 1.173978
(Iteration 1361 / 4900) loss: 1.281196
(Iteration 1371 / 4900) loss: 1.486191
(Iteration 1381 / 4900) loss: 1.184098
(Iteration 1391 / 4900) loss: 1.430220
(Iteration 1401 / 4900) loss: 1.474478
(Iteration 1411 / 4900) loss: 1.542370
(Iteration 1421 / 4900) loss: 1.345799
(Iteration 1431 / 4900) loss: 1.477571
(Iteration 1441 / 4900) loss: 1.449288
(Iteration 1451 / 4900) loss: 1.164258
(Iteration 1461 / 4900) loss: 1.409073
(Epoch 3 / 10) train acc: 0.542000; val_acc: 0.492000
(Iteration 1471 / 4900) loss: 1.377283
(Iteration 1481 / 4900) loss: 1.488322
(Iteration 1491 / 4900) loss: 1.457682
(Iteration 1501 / 4900) loss: 1.279454
(Iteration 1511 / 4900) loss: 1.448994
(Iteration 1521 / 4900) loss: 1.335874
(Iteration 1531 / 4900) loss: 1.669404
(Iteration 1541 / 4900) loss: 1.442587
(Iteration 1551 / 4900) loss: 1.319666
```

```
(Iteration 1561 / 4900) loss: 1.335079
(Iteration 1571 / 4900) loss: 1.419458
(Iteration 1581 / 4900) loss: 1.377134
(Iteration 1591 / 4900) loss: 1.459520
(Iteration 1601 / 4900) loss: 1.450127
(Iteration 1611 / 4900) loss: 1.209540
(Iteration 1621 / 4900) loss: 1.115702
(Iteration 1631 / 4900) loss: 1.376270
(Iteration 1641 / 4900) loss: 1.197735
(Iteration 1651 / 4900) loss: 1.335752
(Iteration 1661 / 4900) loss: 1.362768
(Iteration 1671 / 4900) loss: 1.392151
(Iteration 1681 / 4900) loss: 1.426280
(Iteration 1691 / 4900) loss: 1.337710
(Iteration 1701 / 4900) loss: 1.269451
(Iteration 1711 / 4900) loss: 1.301835
(Iteration 1721 / 4900) loss: 1.235622
(Iteration 1731 / 4900) loss: 1.393686
(Iteration 1741 / 4900) loss: 1.420315
(Iteration 1751 / 4900) loss: 1.435824
(Iteration 1761 / 4900) loss: 1.256546
(Iteration 1771 / 4900) loss: 1.401896
(Iteration 1781 / 4900) loss: 1.352212
(Iteration 1791 / 4900) loss: 1.316108
(Iteration 1801 / 4900) loss: 1.207038
(Iteration 1811 / 4900) loss: 1.341172
(Iteration 1821 / 4900) loss: 1.298591
(Iteration 1831 / 4900) loss: 1.332873
(Iteration 1841 / 4900) loss: 1.327001
(Iteration 1851 / 4900) loss: 1.186589
(Iteration 1861 / 4900) loss: 1.252857
(Iteration 1871 / 4900) loss: 1.461127
(Iteration 1881 / 4900) loss: 1.204893
(Iteration 1891 / 4900) loss: 1.650730
(Iteration 1901 / 4900) loss: 1.245926
(Iteration 1911 / 4900) loss: 1.241789
(Iteration 1921 / 4900) loss: 1.371661
(Iteration 1931 / 4900) loss: 1.452769
(Iteration 1941 / 4900) loss: 1.356564
(Iteration 1951 / 4900) loss: 1.157496
(Epoch 4 / 10) train acc: 0.526000; val_acc: 0.480000
(Iteration 1961 / 4900) loss: 1.564035
(Iteration 1971 / 4900) loss: 1.398725
(Iteration 1981 / 4900) loss: 1.420197
(Iteration 1991 / 4900) loss: 1.193773
(Iteration 2001 / 4900) loss: 1.475411
(Iteration 2011 / 4900) loss: 1.429051
(Iteration 2021 / 4900) loss: 1.176555
```

```
(Iteration 2031 / 4900) loss: 1.172262
(Iteration 2041 / 4900) loss: 1.387955
(Iteration 2051 / 4900) loss: 1.320713
(Iteration 2061 / 4900) loss: 1.420268
(Iteration 2071 / 4900) loss: 1.242452
(Iteration 2081 / 4900) loss: 1.189080
(Iteration 2091 / 4900) loss: 1.218451
(Iteration 2101 / 4900) loss: 1.210292
(Iteration 2111 / 4900) loss: 1.356307
(Iteration 2121 / 4900) loss: 1.220131
(Iteration 2131 / 4900) loss: 1.523703
(Iteration 2141 / 4900) loss: 1.282630
(Iteration 2151 / 4900) loss: 1.310013
(Iteration 2161 / 4900) loss: 1.358017
(Iteration 2171 / 4900) loss: 1.240161
(Iteration 2181 / 4900) loss: 1.077329
(Iteration 2191 / 4900) loss: 1.090974
(Iteration 2201 / 4900) loss: 1.292341
(Iteration 2211 / 4900) loss: 0.984649
(Iteration 2221 / 4900) loss: 1.083718
(Iteration 2231 / 4900) loss: 1.336017
(Iteration 2241 / 4900) loss: 1.201152
(Iteration 2251 / 4900) loss: 1.217495
(Iteration 2261 / 4900) loss: 1.228550
(Iteration 2271 / 4900) loss: 1.297499
(Iteration 2281 / 4900) loss: 1.047241
(Iteration 2291 / 4900) loss: 1.226789
(Iteration 2301 / 4900) loss: 1.342051
(Iteration 2311 / 4900) loss: 1.184695
(Iteration 2321 / 4900) loss: 1.335868
(Iteration 2331 / 4900) loss: 1.578627
(Iteration 2341 / 4900) loss: 1.429272
(Iteration 2351 / 4900) loss: 1.049118
(Iteration 2361 / 4900) loss: 1.508211
(Iteration 2371 / 4900) loss: 1.255842
(Iteration 2381 / 4900) loss: 1.379626
(Iteration 2391 / 4900) loss: 1.241104
(Iteration 2401 / 4900) loss: 1.200491
(Iteration 2411 / 4900) loss: 1.217770
(Iteration 2421 / 4900) loss: 1.226190
(Iteration 2431 / 4900) loss: 1.441494
(Iteration 2441 / 4900) loss: 1.323195
(Epoch 5 / 10) train acc: 0.560000; val_acc: 0.489000
(Iteration 2451 / 4900) loss: 1.190945
(Iteration 2461 / 4900) loss: 1.517051
(Iteration 2471 / 4900) loss: 1.276074
(Iteration 2481 / 4900) loss: 1.171670
(Iteration 2491 / 4900) loss: 1.477165
```

```
(Iteration 2501 / 4900) loss: 1.301164
(Iteration 2511 / 4900) loss: 1.386973
(Iteration 2521 / 4900) loss: 1.201016
(Iteration 2531 / 4900) loss: 1.159647
(Iteration 2541 / 4900) loss: 1.207004
(Iteration 2551 / 4900) loss: 1.248241
(Iteration 2561 / 4900) loss: 1.171593
(Iteration 2571 / 4900) loss: 0.978928
(Iteration 2581 / 4900) loss: 1.124683
(Iteration 2591 / 4900) loss: 1.389014
(Iteration 2601 / 4900) loss: 1.296512
(Iteration 2611 / 4900) loss: 1.274940
(Iteration 2621 / 4900) loss: 1.148744
(Iteration 2631 / 4900) loss: 1.170155
(Iteration 2641 / 4900) loss: 1.246181
(Iteration 2651 / 4900) loss: 1.105728
(Iteration 2661 / 4900) loss: 1.442378
(Iteration 2671 / 4900) loss: 1.197046
(Iteration 2681 / 4900) loss: 1.272836
(Iteration 2691 / 4900) loss: 1.287117
(Iteration 2701 / 4900) loss: 1.278335
(Iteration 2711 / 4900) loss: 1.065855
(Iteration 2721 / 4900) loss: 1.351421
(Iteration 2731 / 4900) loss: 1.325336
(Iteration 2741 / 4900) loss: 1.258774
(Iteration 2751 / 4900) loss: 1.249005
(Iteration 2761 / 4900) loss: 1.164116
(Iteration 2771 / 4900) loss: 1.097477
(Iteration 2781 / 4900) loss: 1.376135
(Iteration 2791 / 4900) loss: 1.152609
(Iteration 2801 / 4900) loss: 1.140888
(Iteration 2811 / 4900) loss: 1.267210
(Iteration 2821 / 4900) loss: 0.992230
(Iteration 2831 / 4900) loss: 1.587677
(Iteration 2841 / 4900) loss: 1.207415
(Iteration 2851 / 4900) loss: 1.210759
(Iteration 2861 / 4900) loss: 1.202820
(Iteration 2871 / 4900) loss: 1.039867
(Iteration 2881 / 4900) loss: 1.144518
(Iteration 2891 / 4900) loss: 1.201646
(Iteration 2901 / 4900) loss: 1.305830
(Iteration 2911 / 4900) loss: 1.080383
(Iteration 2921 / 4900) loss: 1.280845
(Iteration 2931 / 4900) loss: 1.263843
(Epoch 6 / 10) train acc: 0.598000; val_acc: 0.505000
(Iteration 2941 / 4900) loss: 1.208106
(Iteration 2951 / 4900) loss: 1.096441
(Iteration 2961 / 4900) loss: 1.253663
```

```
(Iteration 2971 / 4900) loss: 1.275261
(Iteration 2981 / 4900) loss: 1.295095
(Iteration 2991 / 4900) loss: 1.370957
(Iteration 3001 / 4900) loss: 1.106644
(Iteration 3011 / 4900) loss: 1.113084
(Iteration 3021 / 4900) loss: 1.241873
(Iteration 3031 / 4900) loss: 1.319873
(Iteration 3041 / 4900) loss: 1.295742
(Iteration 3051 / 4900) loss: 1.388698
(Iteration 3061 / 4900) loss: 1.170599
(Iteration 3071 / 4900) loss: 1.292418
(Iteration 3081 / 4900) loss: 1.323230
(Iteration 3091 / 4900) loss: 1.181894
(Iteration 3101 / 4900) loss: 1.298636
(Iteration 3111 / 4900) loss: 1.138499
(Iteration 3121 / 4900) loss: 1.316578
(Iteration 3131 / 4900) loss: 1.124373
(Iteration 3141 / 4900) loss: 1.351506
(Iteration 3151 / 4900) loss: 1.243241
(Iteration 3161 / 4900) loss: 1.071817
(Iteration 3171 / 4900) loss: 1.151646
(Iteration 3181 / 4900) loss: 1.072143
(Iteration 3191 / 4900) loss: 1.230668
(Iteration 3201 / 4900) loss: 1.149538
(Iteration 3211 / 4900) loss: 1.184506
(Iteration 3221 / 4900) loss: 1.083152
(Iteration 3231 / 4900) loss: 1.205005
(Iteration 3241 / 4900) loss: 1.061177
(Iteration 3251 / 4900) loss: 1.192288
(Iteration 3261 / 4900) loss: 1.310534
(Iteration 3271 / 4900) loss: 1.170769
(Iteration 3281 / 4900) loss: 1.136095
(Iteration 3291 / 4900) loss: 1.150452
(Iteration 3301 / 4900) loss: 1.143112
(Iteration 3311 / 4900) loss: 1.222160
(Iteration 3321 / 4900) loss: 1.159329
(Iteration 3331 / 4900) loss: 1.071810
(Iteration 3341 / 4900) loss: 1.251125
(Iteration 3351 / 4900) loss: 1.182925
(Iteration 3361 / 4900) loss: 1.139602
(Iteration 3371 / 4900) loss: 1.243703
(Iteration 3381 / 4900) loss: 1.112990
(Iteration 3391 / 4900) loss: 1.275089
(Iteration 3401 / 4900) loss: 1.164139
(Iteration 3411 / 4900) loss: 0.993305
(Iteration 3421 / 4900) loss: 1.189515
(Epoch 7 / 10) train acc: 0.572000; val_acc: 0.496000
(Iteration 3431 / 4900) loss: 1.280694
```

```
(Iteration 3441 / 4900) loss: 1.351124
(Iteration 3451 / 4900) loss: 1.183911
(Iteration 3461 / 4900) loss: 1.302448
(Iteration 3471 / 4900) loss: 1.136601
(Iteration 3481 / 4900) loss: 1.039843
(Iteration 3491 / 4900) loss: 1.383268
(Iteration 3501 / 4900) loss: 1.181093
(Iteration 3511 / 4900) loss: 1.203892
(Iteration 3521 / 4900) loss: 1.054515
(Iteration 3531 / 4900) loss: 1.067251
(Iteration 3541 / 4900) loss: 1.152965
(Iteration 3551 / 4900) loss: 1.224946
(Iteration 3561 / 4900) loss: 1.186550
(Iteration 3571 / 4900) loss: 0.988629
(Iteration 3581 / 4900) loss: 1.225383
(Iteration 3591 / 4900) loss: 1.078764
(Iteration 3601 / 4900) loss: 1.166868
(Iteration 3611 / 4900) loss: 0.945318
(Iteration 3621 / 4900) loss: 0.990792
(Iteration 3631 / 4900) loss: 1.036656
(Iteration 3641 / 4900) loss: 1.138214
(Iteration 3651 / 4900) loss: 1.129368
(Iteration 3661 / 4900) loss: 1.002369
(Iteration 3671 / 4900) loss: 1.150907
(Iteration 3681 / 4900) loss: 1.122277
(Iteration 3691 / 4900) loss: 1.084253
(Iteration 3701 / 4900) loss: 1.076265
(Iteration 3711 / 4900) loss: 1.098670
(Iteration 3721 / 4900) loss: 1.044704
(Iteration 3731 / 4900) loss: 1.244362
(Iteration 3741 / 4900) loss: 1.162726
(Iteration 3751 / 4900) loss: 1.304333
(Iteration 3761 / 4900) loss: 0.972899
(Iteration 3771 / 4900) loss: 0.957740
(Iteration 3781 / 4900) loss: 1.164640
(Iteration 3791 / 4900) loss: 1.389026
(Iteration 3801 / 4900) loss: 1.064213
(Iteration 3811 / 4900) loss: 1.125896
(Iteration 3821 / 4900) loss: 1.345915
(Iteration 3831 / 4900) loss: 1.102479
(Iteration 3841 / 4900) loss: 1.199508
(Iteration 3851 / 4900) loss: 1.159356
(Iteration 3861 / 4900) loss: 1.081841
(Iteration 3871 / 4900) loss: 1.003576
(Iteration 3881 / 4900) loss: 0.947410
(Iteration 3891 / 4900) loss: 1.171115
(Iteration 3901 / 4900) loss: 1.121216
(Iteration 3911 / 4900) loss: 1.170659
```

```
(Epoch 8 / 10) train acc: 0.590000; val_acc: 0.521000
(Iteration 3921 / 4900) loss: 1.275607
(Iteration 3931 / 4900) loss: 1.177285
(Iteration 3941 / 4900) loss: 1.191995
(Iteration 3951 / 4900) loss: 1.218935
(Iteration 3961 / 4900) loss: 1.288110
(Iteration 3971 / 4900) loss: 1.068049
(Iteration 3981 / 4900) loss: 1.122110
(Iteration 3991 / 4900) loss: 1.280988
(Iteration 4001 / 4900) loss: 1.202297
(Iteration 4011 / 4900) loss: 1.156205
(Iteration 4021 / 4900) loss: 1.119029
(Iteration 4031 / 4900) loss: 0.992425
(Iteration 4041 / 4900) loss: 0.953899
(Iteration 4051 / 4900) loss: 1.113071
(Iteration 4061 / 4900) loss: 1.161857
(Iteration 4071 / 4900) loss: 1.063815
(Iteration 4081 / 4900) loss: 1.005550
(Iteration 4091 / 4900) loss: 1.329143
(Iteration 4101 / 4900) loss: 1.217053
(Iteration 4111 / 4900) loss: 1.333541
(Iteration 4121 / 4900) loss: 1.011238
(Iteration 4131 / 4900) loss: 0.958951
(Iteration 4141 / 4900) loss: 0.941437
(Iteration 4151 / 4900) loss: 1.172191
(Iteration 4161 / 4900) loss: 1.019256
(Iteration 4171 / 4900) loss: 1.070054
(Iteration 4181 / 4900) loss: 1.175176
(Iteration 4191 / 4900) loss: 1.115992
(Iteration 4201 / 4900) loss: 1.068619
(Iteration 4211 / 4900) loss: 0.746802
(Iteration 4221 / 4900) loss: 0.931076
(Iteration 4231 / 4900) loss: 1.159189
(Iteration 4241 / 4900) loss: 1.310605
(Iteration 4251 / 4900) loss: 0.882528
(Iteration 4261 / 4900) loss: 1.116084
(Iteration 4271 / 4900) loss: 1.199217
(Iteration 4281 / 4900) loss: 1.148284
(Iteration 4291 / 4900) loss: 1.061210
(Iteration 4301 / 4900) loss: 1.222963
(Iteration 4311 / 4900) loss: 1.175041
(Iteration 4321 / 4900) loss: 1.297245
(Iteration 4331 / 4900) loss: 1.387637
(Iteration 4341 / 4900) loss: 1.096007
(Iteration 4351 / 4900) loss: 1.068288
(Iteration 4361 / 4900) loss: 1.071341
(Iteration 4371 / 4900) loss: 1.120458
(Iteration 4381 / 4900) loss: 0.888887
```

```
(Iteration 4391 / 4900) loss: 1.229582
(Iteration 4401 / 4900) loss: 1.080848
(Epoch 9 / 10) train acc: 0.654000; val_acc: 0.517000
(Iteration 4411 / 4900) loss: 1.134183
(Iteration 4421 / 4900) loss: 1.089205
(Iteration 4431 / 4900) loss: 1.024752
(Iteration 4441 / 4900) loss: 1.051957
(Iteration 4451 / 4900) loss: 0.984042
(Iteration 4461 / 4900) loss: 1.104598
(Iteration 4471 / 4900) loss: 1.038916
(Iteration 4481 / 4900) loss: 1.059967
(Iteration 4491 / 4900) loss: 0.946605
(Iteration 4501 / 4900) loss: 1.135129
(Iteration 4511 / 4900) loss: 1.042227
(Iteration 4521 / 4900) loss: 1.110811
(Iteration 4531 / 4900) loss: 1.029992
(Iteration 4541 / 4900) loss: 1.105239
(Iteration 4551 / 4900) loss: 1.229575
(Iteration 4561 / 4900) loss: 1.033472
(Iteration 4571 / 4900) loss: 1.041710
(Iteration 4581 / 4900) loss: 1.128547
(Iteration 4591 / 4900) loss: 1.105693
(Iteration 4601 / 4900) loss: 1.097575
(Iteration 4611 / 4900) loss: 1.244507
(Iteration 4621 / 4900) loss: 1.165657
(Iteration 4631 / 4900) loss: 1.044584
(Iteration 4641 / 4900) loss: 1.046535
(Iteration 4651 / 4900) loss: 1.147002
(Iteration 4661 / 4900) loss: 1.127023
(Iteration 4671 / 4900) loss: 1.022780
(Iteration 4681 / 4900) loss: 1.074730
(Iteration 4691 / 4900) loss: 1.050808
(Iteration 4701 / 4900) loss: 1.318061
(Iteration 4711 / 4900) loss: 0.932321
(Iteration 4721 / 4900) loss: 0.899610
(Iteration 4731 / 4900) loss: 1.242382
(Iteration 4741 / 4900) loss: 0.999506
(Iteration 4751 / 4900) loss: 1.144731
(Iteration 4761 / 4900) loss: 0.995633
(Iteration 4771 / 4900) loss: 1.116553
(Iteration 4781 / 4900) loss: 1.192485
(Iteration 4791 / 4900) loss: 1.185316
(Iteration 4801 / 4900) loss: 1.175684
(Iteration 4811 / 4900) loss: 1.200784
(Iteration 4821 / 4900) loss: 0.946305
(Iteration 4831 / 4900) loss: 1.126952
(Iteration 4841 / 4900) loss: 1.037973
(Iteration 4851 / 4900) loss: 1.306330
```

```
(Iteration 4861 / 4900) loss: 1.010534
(Iteration 4871 / 4900) loss: 0.993166
(Iteration 4881 / 4900) loss: 1.076563
(Iteration 4891 / 4900) loss: 1.143884
(Epoch 10 / 10) train acc: 0.640000; val_acc: 0.526000
(Iteration 1 / 4900) loss: 2.302589
(Epoch 0 / 10) train acc: 0.123000; val_acc: 0.147000
(Iteration 11 / 4900) loss: 2.194437
(Iteration 21 / 4900) loss: 2.085598
(Iteration 31 / 4900) loss: 2.090459
(Iteration 41 / 4900) loss: 2.070658
(Iteration 51 / 4900) loss: 2.115776
(Iteration 61 / 4900) loss: 2.034670
(Iteration 71 / 4900) loss: 2.193277
(Iteration 81 / 4900) loss: 2.048801
(Iteration 91 / 4900) loss: 2.050761
(Iteration 101 / 4900) loss: 1.980324
(Iteration 111 / 4900) loss: 1.930651
(Iteration 121 / 4900) loss: 2.008941
(Iteration 131 / 4900) loss: 1.826189
(Iteration 141 / 4900) loss: 1.927577
(Iteration 151 / 4900) loss: 1.973142
(Iteration 161 / 4900) loss: 1.938634
(Iteration 171 / 4900) loss: 1.839348
(Iteration 181 / 4900) loss: 1.965741
(Iteration 191 / 4900) loss: 1.739582
(Iteration 201 / 4900) loss: 1.912974
(Iteration 211 / 4900) loss: 1.910209
(Iteration 221 / 4900) loss: 1.858457
(Iteration 231 / 4900) loss: 1.872569
(Iteration 241 / 4900) loss: 1.999889
(Iteration 251 / 4900) loss: 1.961733
(Iteration 261 / 4900) loss: 1.960875
(Iteration 271 / 4900) loss: 1.813987
(Iteration 281 / 4900) loss: 1.768601
(Iteration 291 / 4900) loss: 1.678482
(Iteration 301 / 4900) loss: 1.765984
(Iteration 311 / 4900) loss: 1.830606
(Iteration 321 / 4900) loss: 1.895216
(Iteration 331 / 4900) loss: 1.847754
(Iteration 341 / 4900) loss: 1.709733
(Iteration 351 / 4900) loss: 1.959937
(Iteration 361 / 4900) loss: 1.970361
(Iteration 371 / 4900) loss: 1.837424
(Iteration 381 / 4900) loss: 1.653473
(Iteration 391 / 4900) loss: 1.851864
(Iteration 401 / 4900) loss: 1.800588
(Iteration 411 / 4900) loss: 1.797296
```

```
(Iteration 421 / 4900) loss: 1.951960
(Iteration 431 / 4900) loss: 1.681954
(Iteration 441 / 4900) loss: 1.733428
(Iteration 451 / 4900) loss: 1.793410
(Iteration 461 / 4900) loss: 1.721676
(Iteration 471 / 4900) loss: 1.798749
(Iteration 481 / 4900) loss: 1.714141
(Epoch 1 / 10) train acc: 0.367000; val_acc: 0.379000
(Iteration 491 / 4900) loss: 1.772318
(Iteration 501 / 4900) loss: 1.635069
(Iteration 511 / 4900) loss: 1.663281
(Iteration 521 / 4900) loss: 1.581785
(Iteration 531 / 4900) loss: 1.705211
(Iteration 541 / 4900) loss: 1.723423
(Iteration 551 / 4900) loss: 1.550469
(Iteration 561 / 4900) loss: 1.588814
(Iteration 571 / 4900) loss: 1.799975
(Iteration 581 / 4900) loss: 1.662504
(Iteration 591 / 4900) loss: 1.600759
(Iteration 601 / 4900) loss: 1.733701
(Iteration 611 / 4900) loss: 1.762063
(Iteration 621 / 4900) loss: 1.736708
(Iteration 631 / 4900) loss: 1.744528
(Iteration 641 / 4900) loss: 1.518223
(Iteration 651 / 4900) loss: 1.709558
(Iteration 661 / 4900) loss: 1.444997
(Iteration 671 / 4900) loss: 1.556678
(Iteration 681 / 4900) loss: 1.600874
(Iteration 691 / 4900) loss: 1.626562
(Iteration 701 / 4900) loss: 1.535633
(Iteration 711 / 4900) loss: 1.431358
(Iteration 721 / 4900) loss: 1.763163
(Iteration 731 / 4900) loss: 1.599616
(Iteration 741 / 4900) loss: 1.625563
(Iteration 751 / 4900) loss: 1.658593
(Iteration 761 / 4900) loss: 1.613284
(Iteration 771 / 4900) loss: 1.755641
(Iteration 781 / 4900) loss: 1.555261
(Iteration 791 / 4900) loss: 1.689832
(Iteration 801 / 4900) loss: 1.577450
(Iteration 811 / 4900) loss: 1.664104
(Iteration 821 / 4900) loss: 1.646993
(Iteration 831 / 4900) loss: 1.379027
(Iteration 841 / 4900) loss: 1.692748
(Iteration 851 / 4900) loss: 1.553626
(Iteration 861 / 4900) loss: 1.478837
(Iteration 871 / 4900) loss: 1.701872
(Iteration 881 / 4900) loss: 1.691440
```

```
(Iteration 891 / 4900) loss: 1.739667
(Iteration 901 / 4900) loss: 1.504301
(Iteration 911 / 4900) loss: 1.652645
(Iteration 921 / 4900) loss: 1.340640
(Iteration 931 / 4900) loss: 1.400894
(Iteration 941 / 4900) loss: 1.536853
(Iteration 951 / 4900) loss: 1.435038
(Iteration 961 / 4900) loss: 1.520310
(Iteration 971 / 4900) loss: 1.384657
(Epoch 2 / 10) train acc: 0.453000; val_acc: 0.429000
(Iteration 981 / 4900) loss: 1.305730
(Iteration 991 / 4900) loss: 1.412624
(Iteration 1001 / 4900) loss: 1.529839
(Iteration 1011 / 4900) loss: 1.617178
(Iteration 1021 / 4900) loss: 1.451350
(Iteration 1031 / 4900) loss: 1.590138
(Iteration 1041 / 4900) loss: 1.473285
(Iteration 1051 / 4900) loss: 1.364385
(Iteration 1061 / 4900) loss: 1.472296
(Iteration 1071 / 4900) loss: 1.370582
(Iteration 1081 / 4900) loss: 1.642445
(Iteration 1091 / 4900) loss: 1.608621
(Iteration 1101 / 4900) loss: 1.719993
(Iteration 1111 / 4900) loss: 1.487527
(Iteration 1121 / 4900) loss: 1.521104
(Iteration 1131 / 4900) loss: 1.598076
(Iteration 1141 / 4900) loss: 1.498862
(Iteration 1151 / 4900) loss: 1.455443
(Iteration 1161 / 4900) loss: 1.569173
(Iteration 1171 / 4900) loss: 1.383136
(Iteration 1181 / 4900) loss: 1.416334
(Iteration 1191 / 4900) loss: 1.562141
(Iteration 1201 / 4900) loss: 1.396267
(Iteration 1211 / 4900) loss: 1.428870
(Iteration 1221 / 4900) loss: 1.567911
(Iteration 1231 / 4900) loss: 1.505138
(Iteration 1241 / 4900) loss: 1.610457
(Iteration 1251 / 4900) loss: 1.824547
(Iteration 1261 / 4900) loss: 1.448892
(Iteration 1271 / 4900) loss: 1.497308
(Iteration 1281 / 4900) loss: 1.339139
(Iteration 1291 / 4900) loss: 1.553871
(Iteration 1301 / 4900) loss: 1.493432
(Iteration 1311 / 4900) loss: 1.524714
(Iteration 1321 / 4900) loss: 1.557944
(Iteration 1331 / 4900) loss: 1.280981
(Iteration 1341 / 4900) loss: 1.531429
(Iteration 1351 / 4900) loss: 1.470178
```

```
(Iteration 1361 / 4900) loss: 1.485951
(Iteration 1371 / 4900) loss: 1.424855
(Iteration 1381 / 4900) loss: 1.376461
(Iteration 1391 / 4900) loss: 1.465866
(Iteration 1401 / 4900) loss: 1.455568
(Iteration 1411 / 4900) loss: 1.313663
(Iteration 1421 / 4900) loss: 1.221209
(Iteration 1431 / 4900) loss: 1.421714
(Iteration 1441 / 4900) loss: 1.461822
(Iteration 1451 / 4900) loss: 1.466280
(Iteration 1461 / 4900) loss: 1.603120
(Epoch 3 / 10) train acc: 0.508000; val_acc: 0.471000
(Iteration 1471 / 4900) loss: 1.322819
(Iteration 1481 / 4900) loss: 1.390151
(Iteration 1491 / 4900) loss: 1.344089
(Iteration 1501 / 4900) loss: 1.454721
(Iteration 1511 / 4900) loss: 1.497325
(Iteration 1521 / 4900) loss: 1.522556
(Iteration 1531 / 4900) loss: 1.342601
(Iteration 1541 / 4900) loss: 1.412845
(Iteration 1551 / 4900) loss: 1.468409
(Iteration 1561 / 4900) loss: 1.389217
(Iteration 1571 / 4900) loss: 1.457340
(Iteration 1581 / 4900) loss: 1.421186
(Iteration 1591 / 4900) loss: 1.369519
(Iteration 1601 / 4900) loss: 1.275277
(Iteration 1611 / 4900) loss: 1.533297
(Iteration 1621 / 4900) loss: 1.592175
(Iteration 1631 / 4900) loss: 1.325319
(Iteration 1641 / 4900) loss: 1.447880
(Iteration 1651 / 4900) loss: 1.346838
(Iteration 1661 / 4900) loss: 1.375114
(Iteration 1671 / 4900) loss: 1.360836
(Iteration 1681 / 4900) loss: 1.497324
(Iteration 1691 / 4900) loss: 1.395743
(Iteration 1701 / 4900) loss: 1.443372
(Iteration 1711 / 4900) loss: 1.425097
(Iteration 1721 / 4900) loss: 1.408875
(Iteration 1731 / 4900) loss: 1.570022
(Iteration 1741 / 4900) loss: 1.409473
(Iteration 1751 / 4900) loss: 1.535766
(Iteration 1761 / 4900) loss: 1.339906
(Iteration 1771 / 4900) loss: 1.456593
(Iteration 1781 / 4900) loss: 1.389934
(Iteration 1791 / 4900) loss: 1.365164
(Iteration 1801 / 4900) loss: 1.393763
(Iteration 1811 / 4900) loss: 1.320515
(Iteration 1821 / 4900) loss: 1.514951
```

```
(Iteration 1831 / 4900) loss: 1.495050
(Iteration 1841 / 4900) loss: 1.426043
(Iteration 1851 / 4900) loss: 1.500194
(Iteration 1861 / 4900) loss: 1.456116
(Iteration 1871 / 4900) loss: 1.369889
(Iteration 1881 / 4900) loss: 1.271375
(Iteration 1891 / 4900) loss: 1.317185
(Iteration 1901 / 4900) loss: 1.526967
(Iteration 1911 / 4900) loss: 1.191393
(Iteration 1921 / 4900) loss: 1.596057
(Iteration 1931 / 4900) loss: 1.510331
(Iteration 1941 / 4900) loss: 1.282599
(Iteration 1951 / 4900) loss: 1.493128
(Epoch 4 / 10) train acc: 0.519000; val_acc: 0.486000
(Iteration 1961 / 4900) loss: 1.513809
(Iteration 1971 / 4900) loss: 1.212792
(Iteration 1981 / 4900) loss: 1.249352
(Iteration 1991 / 4900) loss: 1.444478
(Iteration 2001 / 4900) loss: 1.302749
(Iteration 2011 / 4900) loss: 1.415354
(Iteration 2021 / 4900) loss: 1.401847
(Iteration 2031 / 4900) loss: 1.340380
(Iteration 2041 / 4900) loss: 1.313161
(Iteration 2051 / 4900) loss: 1.441622
(Iteration 2061 / 4900) loss: 1.017505
(Iteration 2071 / 4900) loss: 1.385313
(Iteration 2081 / 4900) loss: 1.347206
(Iteration 2091 / 4900) loss: 1.258968
(Iteration 2101 / 4900) loss: 1.269057
(Iteration 2111 / 4900) loss: 1.235555
(Iteration 2121 / 4900) loss: 1.368433
(Iteration 2131 / 4900) loss: 1.391276
(Iteration 2141 / 4900) loss: 1.314158
(Iteration 2151 / 4900) loss: 1.297020
(Iteration 2161 / 4900) loss: 1.478157
(Iteration 2171 / 4900) loss: 1.012776
(Iteration 2181 / 4900) loss: 1.448263
(Iteration 2191 / 4900) loss: 1.309649
(Iteration 2201 / 4900) loss: 1.463598
(Iteration 2211 / 4900) loss: 1.400072
(Iteration 2221 / 4900) loss: 1.427478
(Iteration 2231 / 4900) loss: 1.375382
(Iteration 2241 / 4900) loss: 1.264383
(Iteration 2251 / 4900) loss: 1.342856
(Iteration 2261 / 4900) loss: 1.373292
(Iteration 2271 / 4900) loss: 1.232070
(Iteration 2281 / 4900) loss: 1.255944
(Iteration 2291 / 4900) loss: 1.501623
```

```
(Iteration 2301 / 4900) loss: 1.303073
(Iteration 2311 / 4900) loss: 1.201223
(Iteration 2321 / 4900) loss: 1.362290
(Iteration 2331 / 4900) loss: 1.508956
(Iteration 2341 / 4900) loss: 1.310495
(Iteration 2351 / 4900) loss: 1.337597
(Iteration 2361 / 4900) loss: 1.305912
(Iteration 2371 / 4900) loss: 1.395237
(Iteration 2381 / 4900) loss: 1.396774
(Iteration 2391 / 4900) loss: 1.268717
(Iteration 2401 / 4900) loss: 1.227474
(Iteration 2411 / 4900) loss: 1.450874
(Iteration 2421 / 4900) loss: 1.423981
(Iteration 2431 / 4900) loss: 1.383338
(Iteration 2441 / 4900) loss: 1.668977
(Epoch 5 / 10) train acc: 0.528000; val_acc: 0.489000
(Iteration 2451 / 4900) loss: 1.134723
(Iteration 2461 / 4900) loss: 1.356382
(Iteration 2471 / 4900) loss: 1.322877
(Iteration 2481 / 4900) loss: 1.281176
(Iteration 2491 / 4900) loss: 1.378147
(Iteration 2501 / 4900) loss: 1.360564
(Iteration 2511 / 4900) loss: 1.220150
(Iteration 2521 / 4900) loss: 1.327916
(Iteration 2531 / 4900) loss: 1.415223
(Iteration 2541 / 4900) loss: 1.179699
(Iteration 2551 / 4900) loss: 1.288804
(Iteration 2561 / 4900) loss: 1.281759
(Iteration 2571 / 4900) loss: 1.272876
(Iteration 2581 / 4900) loss: 1.136188
(Iteration 2591 / 4900) loss: 1.436203
(Iteration 2601 / 4900) loss: 1.336522
(Iteration 2611 / 4900) loss: 1.273195
(Iteration 2621 / 4900) loss: 1.213532
(Iteration 2631 / 4900) loss: 1.346500
(Iteration 2641 / 4900) loss: 1.533636
(Iteration 2651 / 4900) loss: 1.246646
(Iteration 2661 / 4900) loss: 1.284641
(Iteration 2671 / 4900) loss: 1.326895
(Iteration 2681 / 4900) loss: 1.220137
(Iteration 2691 / 4900) loss: 1.243492
(Iteration 2701 / 4900) loss: 1.285178
(Iteration 2711 / 4900) loss: 1.197416
(Iteration 2721 / 4900) loss: 1.443180
(Iteration 2731 / 4900) loss: 1.242280
(Iteration 2741 / 4900) loss: 1.174176
(Iteration 2751 / 4900) loss: 1.132543
(Iteration 2761 / 4900) loss: 1.345073
```

```
(Iteration 2771 / 4900) loss: 1.111864
(Iteration 2781 / 4900) loss: 1.374623
(Iteration 2791 / 4900) loss: 1.182959
(Iteration 2801 / 4900) loss: 1.201193
(Iteration 2811 / 4900) loss: 1.349404
(Iteration 2821 / 4900) loss: 1.525206
(Iteration 2831 / 4900) loss: 1.339878
(Iteration 2841 / 4900) loss: 1.141098
(Iteration 2851 / 4900) loss: 1.297836
(Iteration 2861 / 4900) loss: 1.337622
(Iteration 2871 / 4900) loss: 1.199402
(Iteration 2881 / 4900) loss: 1.108325
(Iteration 2891 / 4900) loss: 1.139896
(Iteration 2901 / 4900) loss: 1.236940
(Iteration 2911 / 4900) loss: 1.183771
(Iteration 2921 / 4900) loss: 1.232938
(Iteration 2931 / 4900) loss: 1.426939
(Epoch 6 / 10) train acc: 0.537000; val_acc: 0.516000
(Iteration 2941 / 4900) loss: 1.199303
(Iteration 2951 / 4900) loss: 1.215875
(Iteration 2961 / 4900) loss: 1.105118
(Iteration 2971 / 4900) loss: 1.126223
(Iteration 2981 / 4900) loss: 1.339721
(Iteration 2991 / 4900) loss: 1.244890
(Iteration 3001 / 4900) loss: 1.323190
(Iteration 3011 / 4900) loss: 1.206402
(Iteration 3021 / 4900) loss: 1.172883
(Iteration 3031 / 4900) loss: 1.370422
(Iteration 3041 / 4900) loss: 1.325042
(Iteration 3051 / 4900) loss: 1.436420
(Iteration 3061 / 4900) loss: 1.511235
(Iteration 3071 / 4900) loss: 1.321079
(Iteration 3081 / 4900) loss: 1.063655
(Iteration 3091 / 4900) loss: 1.359040
(Iteration 3101 / 4900) loss: 1.082211
(Iteration 3111 / 4900) loss: 1.271377
(Iteration 3121 / 4900) loss: 1.331674
(Iteration 3131 / 4900) loss: 1.256430
(Iteration 3141 / 4900) loss: 1.089730
(Iteration 3151 / 4900) loss: 0.979873
(Iteration 3161 / 4900) loss: 1.099808
(Iteration 3171 / 4900) loss: 1.276883
(Iteration 3181 / 4900) loss: 1.142018
(Iteration 3191 / 4900) loss: 1.171434
(Iteration 3201 / 4900) loss: 1.347122
(Iteration 3211 / 4900) loss: 1.443657
(Iteration 3221 / 4900) loss: 1.412396
(Iteration 3231 / 4900) loss: 1.179133
```

```
(Iteration 3241 / 4900) loss: 1.217247
(Iteration 3251 / 4900) loss: 1.285052
(Iteration 3261 / 4900) loss: 0.999646
(Iteration 3271 / 4900) loss: 1.240487
(Iteration 3281 / 4900) loss: 1.259220
(Iteration 3291 / 4900) loss: 1.339694
(Iteration 3301 / 4900) loss: 1.213899
(Iteration 3311 / 4900) loss: 1.074508
(Iteration 3321 / 4900) loss: 1.351990
(Iteration 3331 / 4900) loss: 1.216999
(Iteration 3341 / 4900) loss: 1.133921
(Iteration 3351 / 4900) loss: 1.317737
(Iteration 3361 / 4900) loss: 1.333263
(Iteration 3371 / 4900) loss: 1.097524
(Iteration 3381 / 4900) loss: 1.212938
(Iteration 3391 / 4900) loss: 1.187424
(Iteration 3401 / 4900) loss: 1.288679
(Iteration 3411 / 4900) loss: 1.136824
(Iteration 3421 / 4900) loss: 1.193981
(Epoch 7 / 10) train acc: 0.557000; val_acc: 0.511000
(Iteration 3431 / 4900) loss: 1.148082
(Iteration 3441 / 4900) loss: 1.196158
(Iteration 3451 / 4900) loss: 1.394854
(Iteration 3461 / 4900) loss: 1.299593
(Iteration 3471 / 4900) loss: 1.281818
(Iteration 3481 / 4900) loss: 1.265756
(Iteration 3491 / 4900) loss: 1.222609
(Iteration 3501 / 4900) loss: 1.213689
(Iteration 3511 / 4900) loss: 1.198230
(Iteration 3521 / 4900) loss: 1.270597
(Iteration 3531 / 4900) loss: 1.187297
(Iteration 3541 / 4900) loss: 1.098947
(Iteration 3551 / 4900) loss: 1.205257
(Iteration 3561 / 4900) loss: 1.153891
(Iteration 3571 / 4900) loss: 1.349992
(Iteration 3581 / 4900) loss: 1.156234
(Iteration 3591 / 4900) loss: 1.259826
(Iteration 3601 / 4900) loss: 1.212274
(Iteration 3611 / 4900) loss: 1.201996
(Iteration 3621 / 4900) loss: 1.257326
(Iteration 3631 / 4900) loss: 1.256500
(Iteration 3641 / 4900) loss: 1.273892
(Iteration 3651 / 4900) loss: 1.215130
(Iteration 3661 / 4900) loss: 1.021761
(Iteration 3671 / 4900) loss: 1.493381
(Iteration 3681 / 4900) loss: 1.295149
(Iteration 3691 / 4900) loss: 1.224384
(Iteration 3701 / 4900) loss: 1.235182
```

```
(Iteration 3711 / 4900) loss: 1.429723
(Iteration 3721 / 4900) loss: 1.121962
(Iteration 3731 / 4900) loss: 1.298983
(Iteration 3741 / 4900) loss: 1.090710
(Iteration 3751 / 4900) loss: 1.151754
(Iteration 3761 / 4900) loss: 1.025216
(Iteration 3771 / 4900) loss: 1.218897
(Iteration 3781 / 4900) loss: 1.323828
(Iteration 3791 / 4900) loss: 1.264866
(Iteration 3801 / 4900) loss: 1.250219
(Iteration 3811 / 4900) loss: 1.225886
(Iteration 3821 / 4900) loss: 1.296205
(Iteration 3831 / 4900) loss: 1.313196
(Iteration 3841 / 4900) loss: 1.260735
(Iteration 3851 / 4900) loss: 1.240909
(Iteration 3861 / 4900) loss: 1.433392
(Iteration 3871 / 4900) loss: 1.271344
(Iteration 3881 / 4900) loss: 1.022745
(Iteration 3891 / 4900) loss: 1.090586
(Iteration 3901 / 4900) loss: 0.995883
(Iteration 3911 / 4900) loss: 1.288873
(Epoch 8 / 10) train acc: 0.567000; val_acc: 0.529000
(Iteration 3921 / 4900) loss: 1.233960
(Iteration 3931 / 4900) loss: 1.265602
(Iteration 3941 / 4900) loss: 1.235856
(Iteration 3951 / 4900) loss: 1.273995
(Iteration 3961 / 4900) loss: 1.265941
(Iteration 3971 / 4900) loss: 1.285178
(Iteration 3981 / 4900) loss: 1.106113
(Iteration 3991 / 4900) loss: 1.026785
(Iteration 4001 / 4900) loss: 1.326731
(Iteration 4011 / 4900) loss: 0.971264
(Iteration 4021 / 4900) loss: 0.983071
(Iteration 4031 / 4900) loss: 1.185713
(Iteration 4041 / 4900) loss: 1.206618
(Iteration 4051 / 4900) loss: 1.231678
(Iteration 4061 / 4900) loss: 1.072480
(Iteration 4071 / 4900) loss: 1.245397
(Iteration 4081 / 4900) loss: 1.130160
(Iteration 4091 / 4900) loss: 1.043999
(Iteration 4101 / 4900) loss: 1.280971
(Iteration 4111 / 4900) loss: 1.221232
(Iteration 4121 / 4900) loss: 1.283848
(Iteration 4131 / 4900) loss: 1.267026
(Iteration 4141 / 4900) loss: 1.112030
(Iteration 4151 / 4900) loss: 1.224835
(Iteration 4161 / 4900) loss: 1.107728
(Iteration 4171 / 4900) loss: 1.056872
```

```
(Iteration 4181 / 4900) loss: 1.110688
(Iteration 4191 / 4900) loss: 1.224276
(Iteration 4201 / 4900) loss: 1.196065
(Iteration 4211 / 4900) loss: 1.188136
(Iteration 4221 / 4900) loss: 1.097619
(Iteration 4231 / 4900) loss: 1.096093
(Iteration 4241 / 4900) loss: 1.196831
(Iteration 4251 / 4900) loss: 1.200063
(Iteration 4261 / 4900) loss: 1.168270
(Iteration 4271 / 4900) loss: 1.072223
(Iteration 4281 / 4900) loss: 1.089094
(Iteration 4291 / 4900) loss: 1.053657
(Iteration 4301 / 4900) loss: 1.060682
(Iteration 4311 / 4900) loss: 1.231129
(Iteration 4321 / 4900) loss: 1.052836
(Iteration 4331 / 4900) loss: 1.123148
(Iteration 4341 / 4900) loss: 1.251623
(Iteration 4351 / 4900) loss: 1.141528
(Iteration 4361 / 4900) loss: 1.033416
(Iteration 4371 / 4900) loss: 1.204617
(Iteration 4381 / 4900) loss: 1.257439
(Iteration 4391 / 4900) loss: 1.030975
(Iteration 4401 / 4900) loss: 1.094690
(Epoch 9 / 10) train acc: 0.619000; val_acc: 0.524000
(Iteration 4411 / 4900) loss: 1.147383
(Iteration 4421 / 4900) loss: 1.178666
(Iteration 4431 / 4900) loss: 1.159735
(Iteration 4441 / 4900) loss: 1.100325
(Iteration 4451 / 4900) loss: 1.288332
(Iteration 4461 / 4900) loss: 1.292665
(Iteration 4471 / 4900) loss: 1.241672
(Iteration 4481 / 4900) loss: 1.398294
(Iteration 4491 / 4900) loss: 1.253244
(Iteration 4501 / 4900) loss: 1.048624
(Iteration 4511 / 4900) loss: 1.132486
(Iteration 4521 / 4900) loss: 1.206180
(Iteration 4531 / 4900) loss: 1.072940
(Iteration 4541 / 4900) loss: 1.041014
(Iteration 4551 / 4900) loss: 1.136138
(Iteration 4561 / 4900) loss: 0.923467
(Iteration 4571 / 4900) loss: 1.138195
(Iteration 4581 / 4900) loss: 1.051389
(Iteration 4591 / 4900) loss: 1.083276
(Iteration 4601 / 4900) loss: 1.104043
(Iteration 4611 / 4900) loss: 1.151195
(Iteration 4621 / 4900) loss: 1.097349
(Iteration 4631 / 4900) loss: 1.103613
(Iteration 4641 / 4900) loss: 1.128503
```

```
(Iteration 4651 / 4900) loss: 1.442094
(Iteration 4661 / 4900) loss: 1.344291
(Iteration 4671 / 4900) loss: 1.109816
(Iteration 4681 / 4900) loss: 1.123508
(Iteration 4691 / 4900) loss: 1.272841
(Iteration 4701 / 4900) loss: 1.187343
(Iteration 4711 / 4900) loss: 1.194403
(Iteration 4721 / 4900) loss: 1.242341
(Iteration 4731 / 4900) loss: 1.099347
(Iteration 4741 / 4900) loss: 1.183917
(Iteration 4751 / 4900) loss: 1.153171
(Iteration 4761 / 4900) loss: 1.071652
(Iteration 4771 / 4900) loss: 1.091626
(Iteration 4781 / 4900) loss: 1.016561
(Iteration 4791 / 4900) loss: 0.881535
(Iteration 4801 / 4900) loss: 1.247007
(Iteration 4811 / 4900) loss: 1.199783
(Iteration 4821 / 4900) loss: 0.974229
(Iteration 4831 / 4900) loss: 1.160697
(Iteration 4841 / 4900) loss: 1.060314
(Iteration 4851 / 4900) loss: 1.205595
(Iteration 4861 / 4900) loss: 0.975251
(Iteration 4871 / 4900) loss: 1.061253
(Iteration 4881 / 4900) loss: 1.131747
(Iteration 4891 / 4900) loss: 1.198830
(Epoch 10 / 10) train acc: 0.591000; val_acc: 0.529000
(Iteration 1 / 4900) loss: 2.761459
(Epoch 0 / 10) train acc: 0.126000; val_acc: 0.151000
(Iteration 11 / 4900) loss: 2.149932
(Iteration 21 / 4900) loss: 1.980632
(Iteration 31 / 4900) loss: 2.147385
(Iteration 41 / 4900) loss: 1.902043
(Iteration 51 / 4900) loss: 2.033849
(Iteration 61 / 4900) loss: 1.842715
(Iteration 71 / 4900) loss: 1.675006
(Iteration 81 / 4900) loss: 1.986584
(Iteration 91 / 4900) loss: 1.857502
(Iteration 101 / 4900) loss: 1.672985
(Iteration 111 / 4900) loss: 1.656188
(Iteration 121 / 4900) loss: 1.723892
(Iteration 131 / 4900) loss: 1.804278
(Iteration 141 / 4900) loss: 1.809617
(Iteration 151 / 4900) loss: 1.724903
(Iteration 161 / 4900) loss: 1.663858
(Iteration 171 / 4900) loss: 1.795287
(Iteration 181 / 4900) loss: 1.632869
(Iteration 191 / 4900) loss: 1.683435
(Iteration 201 / 4900) loss: 1.689048
```

```
(Iteration 211 / 4900) loss: 1.600030
(Iteration 221 / 4900) loss: 1.890354
(Iteration 231 / 4900) loss: 1.639920
(Iteration 241 / 4900) loss: 1.706366
(Iteration 251 / 4900) loss: 1.554974
(Iteration 261 / 4900) loss: 1.707594
(Iteration 271 / 4900) loss: 1.559846
(Iteration 281 / 4900) loss: 1.662530
(Iteration 291 / 4900) loss: 1.727323
(Iteration 301 / 4900) loss: 1.834312
(Iteration 311 / 4900) loss: 1.646183
(Iteration 321 / 4900) loss: 1.754840
(Iteration 331 / 4900) loss: 1.654447
(Iteration 341 / 4900) loss: 1.717101
(Iteration 351 / 4900) loss: 1.444837
(Iteration 361 / 4900) loss: 1.496262
(Iteration 371 / 4900) loss: 1.477530
(Iteration 381 / 4900) loss: 1.462705
(Iteration 391 / 4900) loss: 1.549180
(Iteration 401 / 4900) loss: 1.574998
(Iteration 411 / 4900) loss: 1.530813
(Iteration 421 / 4900) loss: 1.643045
(Iteration 431 / 4900) loss: 1.596253
(Iteration 441 / 4900) loss: 1.685628
(Iteration 451 / 4900) loss: 1.742051
(Iteration 461 / 4900) loss: 1.575848
(Iteration 471 / 4900) loss: 1.605868
(Iteration 481 / 4900) loss: 1.576994
(Epoch 1 / 10) train acc: 0.441000; val_acc: 0.408000
(Iteration 491 / 4900) loss: 1.723557
(Iteration 501 / 4900) loss: 1.667105
(Iteration 511 / 4900) loss: 1.452533
(Iteration 521 / 4900) loss: 1.595586
(Iteration 531 / 4900) loss: 1.320919
(Iteration 541 / 4900) loss: 1.675133
(Iteration 551 / 4900) loss: 1.503081
(Iteration 561 / 4900) loss: 1.563664
(Iteration 571 / 4900) loss: 1.391650
(Iteration 581 / 4900) loss: 1.638128
(Iteration 591 / 4900) loss: 1.548086
(Iteration 601 / 4900) loss: 1.537016
(Iteration 611 / 4900) loss: 1.873626
(Iteration 621 / 4900) loss: 1.490093
(Iteration 631 / 4900) loss: 1.623571
(Iteration 641 / 4900) loss: 1.482724
(Iteration 651 / 4900) loss: 1.446952
(Iteration 661 / 4900) loss: 1.649301
(Iteration 671 / 4900) loss: 1.507629
```

```
(Iteration 681 / 4900) loss: 1.359154
(Iteration 691 / 4900) loss: 1.394263
(Iteration 701 / 4900) loss: 1.591268
(Iteration 711 / 4900) loss: 1.434752
(Iteration 721 / 4900) loss: 1.385134
(Iteration 731 / 4900) loss: 1.290435
(Iteration 741 / 4900) loss: 1.505414
(Iteration 751 / 4900) loss: 1.533736
(Iteration 761 / 4900) loss: 1.609938
(Iteration 771 / 4900) loss: 1.499110
(Iteration 781 / 4900) loss: 1.469011
(Iteration 791 / 4900) loss: 1.327738
(Iteration 801 / 4900) loss: 1.433466
(Iteration 811 / 4900) loss: 1.389319
(Iteration 821 / 4900) loss: 1.185313
(Iteration 831 / 4900) loss: 1.589481
(Iteration 841 / 4900) loss: 1.584245
(Iteration 851 / 4900) loss: 1.398156
(Iteration 861 / 4900) loss: 1.337001
(Iteration 871 / 4900) loss: 1.601588
(Iteration 881 / 4900) loss: 1.423942
(Iteration 891 / 4900) loss: 1.472407
(Iteration 901 / 4900) loss: 1.357556
(Iteration 911 / 4900) loss: 1.547071
(Iteration 921 / 4900) loss: 1.394580
(Iteration 931 / 4900) loss: 1.562595
(Iteration 941 / 4900) loss: 1.530991
(Iteration 951 / 4900) loss: 1.372097
(Iteration 961 / 4900) loss: 1.336840
(Iteration 971 / 4900) loss: 1.528707
(Epoch 2 / 10) train acc: 0.474000; val_acc: 0.462000
(Iteration 981 / 4900) loss: 1.465373
(Iteration 991 / 4900) loss: 1.292498
(Iteration 1001 / 4900) loss: 1.357871
(Iteration 1011 / 4900) loss: 1.443520
(Iteration 1021 / 4900) loss: 1.464635
(Iteration 1031 / 4900) loss: 1.451140
(Iteration 1041 / 4900) loss: 1.462433
(Iteration 1051 / 4900) loss: 1.449928
(Iteration 1061 / 4900) loss: 1.217557
(Iteration 1071 / 4900) loss: 1.440502
(Iteration 1081 / 4900) loss: 1.422502
(Iteration 1091 / 4900) loss: 1.417078
(Iteration 1101 / 4900) loss: 1.380013
(Iteration 1111 / 4900) loss: 1.414392
(Iteration 1121 / 4900) loss: 1.257800
(Iteration 1131 / 4900) loss: 1.357137
(Iteration 1141 / 4900) loss: 1.608742
```

```
(Iteration 1151 / 4900) loss: 1.442756
(Iteration 1161 / 4900) loss: 1.451462
(Iteration 1171 / 4900) loss: 1.432177
(Iteration 1181 / 4900) loss: 1.415128
(Iteration 1191 / 4900) loss: 1.464234
(Iteration 1201 / 4900) loss: 1.437650
(Iteration 1211 / 4900) loss: 1.403049
(Iteration 1221 / 4900) loss: 1.405863
(Iteration 1231 / 4900) loss: 1.492702
(Iteration 1241 / 4900) loss: 1.582427
(Iteration 1251 / 4900) loss: 1.382965
(Iteration 1261 / 4900) loss: 1.478273
(Iteration 1271 / 4900) loss: 1.499371
(Iteration 1281 / 4900) loss: 1.397566
(Iteration 1291 / 4900) loss: 1.340547
(Iteration 1301 / 4900) loss: 1.486083
(Iteration 1311 / 4900) loss: 1.451382
(Iteration 1321 / 4900) loss: 1.572165
(Iteration 1331 / 4900) loss: 1.327669
(Iteration 1341 / 4900) loss: 1.344530
(Iteration 1351 / 4900) loss: 1.548601
(Iteration 1361 / 4900) loss: 1.430083
(Iteration 1371 / 4900) loss: 1.390143
(Iteration 1381 / 4900) loss: 1.397539
(Iteration 1391 / 4900) loss: 1.458934
(Iteration 1401 / 4900) loss: 1.279333
(Iteration 1411 / 4900) loss: 1.308037
(Iteration 1421 / 4900) loss: 1.228003
(Iteration 1431 / 4900) loss: 1.423409
(Iteration 1441 / 4900) loss: 1.542734
(Iteration 1451 / 4900) loss: 1.378142
(Iteration 1461 / 4900) loss: 1.406688
(Epoch 3 / 10) train acc: 0.515000; val_acc: 0.462000
(Iteration 1471 / 4900) loss: 1.358227
(Iteration 1481 / 4900) loss: 1.275692
(Iteration 1491 / 4900) loss: 1.285282
(Iteration 1501 / 4900) loss: 1.335826
(Iteration 1511 / 4900) loss: 1.358467
(Iteration 1521 / 4900) loss: 1.454649
(Iteration 1531 / 4900) loss: 1.506035
(Iteration 1541 / 4900) loss: 1.516626
(Iteration 1551 / 4900) loss: 1.250088
(Iteration 1561 / 4900) loss: 1.344221
(Iteration 1571 / 4900) loss: 1.287122
(Iteration 1581 / 4900) loss: 1.533653
(Iteration 1591 / 4900) loss: 1.316438
(Iteration 1601 / 4900) loss: 1.359754
(Iteration 1611 / 4900) loss: 1.411369
```

```
(Iteration 1621 / 4900) loss: 1.309111
(Iteration 1631 / 4900) loss: 1.349502
(Iteration 1641 / 4900) loss: 1.308794
(Iteration 1651 / 4900) loss: 1.250282
(Iteration 1661 / 4900) loss: 1.225942
(Iteration 1671 / 4900) loss: 1.455747
(Iteration 1681 / 4900) loss: 1.303581
(Iteration 1691 / 4900) loss: 1.310209
(Iteration 1701 / 4900) loss: 1.471696
(Iteration 1711 / 4900) loss: 1.440533
(Iteration 1721 / 4900) loss: 1.361456
(Iteration 1731 / 4900) loss: 1.382624
(Iteration 1741 / 4900) loss: 1.583427
(Iteration 1751 / 4900) loss: 1.503644
(Iteration 1761 / 4900) loss: 1.380454
(Iteration 1771 / 4900) loss: 1.357969
(Iteration 1781 / 4900) loss: 1.139575
(Iteration 1791 / 4900) loss: 1.361850
(Iteration 1801 / 4900) loss: 1.262860
(Iteration 1811 / 4900) loss: 1.328533
(Iteration 1821 / 4900) loss: 1.483874
(Iteration 1831 / 4900) loss: 1.488904
(Iteration 1841 / 4900) loss: 1.592786
(Iteration 1851 / 4900) loss: 1.257699
(Iteration 1861 / 4900) loss: 1.248197
(Iteration 1871 / 4900) loss: 1.254161
(Iteration 1881 / 4900) loss: 1.498558
(Iteration 1891 / 4900) loss: 1.414874
(Iteration 1901 / 4900) loss: 1.356702
(Iteration 1911 / 4900) loss: 1.210806
(Iteration 1921 / 4900) loss: 1.485925
(Iteration 1931 / 4900) loss: 1.313209
(Iteration 1941 / 4900) loss: 1.194842
(Iteration 1951 / 4900) loss: 1.281003
(Epoch 4 / 10) train acc: 0.534000; val_acc: 0.471000
(Iteration 1961 / 4900) loss: 1.370732
(Iteration 1971 / 4900) loss: 1.369159
(Iteration 1981 / 4900) loss: 1.463638
(Iteration 1991 / 4900) loss: 1.337256
(Iteration 2001 / 4900) loss: 1.231043
(Iteration 2011 / 4900) loss: 1.442135
(Iteration 2021 / 4900) loss: 1.423155
(Iteration 2031 / 4900) loss: 1.378476
(Iteration 2041 / 4900) loss: 1.290117
(Iteration 2051 / 4900) loss: 1.269232
(Iteration 2061 / 4900) loss: 1.503637
(Iteration 2071 / 4900) loss: 1.318380
(Iteration 2081 / 4900) loss: 1.353239
```

```
(Iteration 2091 / 4900) loss: 1.345945
(Iteration 2101 / 4900) loss: 1.337929
(Iteration 2111 / 4900) loss: 1.176395
(Iteration 2121 / 4900) loss: 1.416763
(Iteration 2131 / 4900) loss: 1.370023
(Iteration 2141 / 4900) loss: 1.491637
(Iteration 2151 / 4900) loss: 1.366690
(Iteration 2161 / 4900) loss: 1.510616
(Iteration 2171 / 4900) loss: 1.431747
(Iteration 2181 / 4900) loss: 1.358476
(Iteration 2191 / 4900) loss: 1.172320
(Iteration 2201 / 4900) loss: 1.265290
(Iteration 2211 / 4900) loss: 1.351398
(Iteration 2221 / 4900) loss: 1.306144
(Iteration 2231 / 4900) loss: 1.459891
(Iteration 2241 / 4900) loss: 1.324579
(Iteration 2251 / 4900) loss: 1.352485
(Iteration 2261 / 4900) loss: 1.574648
(Iteration 2271 / 4900) loss: 1.249057
(Iteration 2281 / 4900) loss: 1.280560
(Iteration 2291 / 4900) loss: 1.264560
(Iteration 2301 / 4900) loss: 1.091939
(Iteration 2311 / 4900) loss: 1.305499
(Iteration 2321 / 4900) loss: 1.273942
(Iteration 2331 / 4900) loss: 1.200701
(Iteration 2341 / 4900) loss: 1.241480
(Iteration 2351 / 4900) loss: 1.079018
(Iteration 2361 / 4900) loss: 1.299275
(Iteration 2371 / 4900) loss: 1.259672
(Iteration 2381 / 4900) loss: 1.223012
(Iteration 2391 / 4900) loss: 1.178522
(Iteration 2401 / 4900) loss: 1.426185
(Iteration 2411 / 4900) loss: 1.162240
(Iteration 2421 / 4900) loss: 1.258414
(Iteration 2431 / 4900) loss: 1.214009
(Iteration 2441 / 4900) loss: 1.394486
(Epoch 5 / 10) train acc: 0.553000; val_acc: 0.475000
(Iteration 2451 / 4900) loss: 1.354680
(Iteration 2461 / 4900) loss: 1.330800
(Iteration 2471 / 4900) loss: 1.049693
(Iteration 2481 / 4900) loss: 1.535231
(Iteration 2491 / 4900) loss: 1.168017
(Iteration 2501 / 4900) loss: 1.348500
(Iteration 2511 / 4900) loss: 1.367137
(Iteration 2521 / 4900) loss: 1.286371
(Iteration 2531 / 4900) loss: 1.361647
(Iteration 2541 / 4900) loss: 1.283686
(Iteration 2551 / 4900) loss: 1.343030
```

```
(Iteration 2561 / 4900) loss: 1.219143
(Iteration 2571 / 4900) loss: 1.206262
(Iteration 2581 / 4900) loss: 1.068126
(Iteration 2591 / 4900) loss: 1.301322
(Iteration 2601 / 4900) loss: 1.256469
(Iteration 2611 / 4900) loss: 1.301044
(Iteration 2621 / 4900) loss: 1.359948
(Iteration 2631 / 4900) loss: 1.138806
(Iteration 2641 / 4900) loss: 1.362159
(Iteration 2651 / 4900) loss: 1.238684
(Iteration 2661 / 4900) loss: 1.333467
(Iteration 2671 / 4900) loss: 1.336555
(Iteration 2681 / 4900) loss: 1.168592
(Iteration 2691 / 4900) loss: 1.286590
(Iteration 2701 / 4900) loss: 1.392946
(Iteration 2711 / 4900) loss: 1.331038
(Iteration 2721 / 4900) loss: 1.252754
(Iteration 2731 / 4900) loss: 1.202285
(Iteration 2741 / 4900) loss: 1.433371
(Iteration 2751 / 4900) loss: 1.423723
(Iteration 2761 / 4900) loss: 1.156913
(Iteration 2771 / 4900) loss: 1.161256
(Iteration 2781 / 4900) loss: 1.245535
(Iteration 2791 / 4900) loss: 1.138069
(Iteration 2801 / 4900) loss: 1.166447
(Iteration 2811 / 4900) loss: 1.274430
(Iteration 2821 / 4900) loss: 1.415053
(Iteration 2831 / 4900) loss: 1.449447
(Iteration 2841 / 4900) loss: 1.231246
(Iteration 2851 / 4900) loss: 1.404443
(Iteration 2861 / 4900) loss: 1.300779
(Iteration 2871 / 4900) loss: 1.375723
(Iteration 2881 / 4900) loss: 1.173522
(Iteration 2891 / 4900) loss: 1.417742
(Iteration 2901 / 4900) loss: 1.071994
(Iteration 2911 / 4900) loss: 1.277562
(Iteration 2921 / 4900) loss: 1.125805
(Iteration 2931 / 4900) loss: 1.180849
(Epoch 6 / 10) train acc: 0.549000; val_acc: 0.496000
(Iteration 2941 / 4900) loss: 1.317995
(Iteration 2951 / 4900) loss: 1.244278
(Iteration 2961 / 4900) loss: 1.205765
(Iteration 2971 / 4900) loss: 1.299457
(Iteration 2981 / 4900) loss: 1.086135
(Iteration 2991 / 4900) loss: 1.150392
(Iteration 3001 / 4900) loss: 1.226575
(Iteration 3011 / 4900) loss: 1.251710
(Iteration 3021 / 4900) loss: 1.167526
```

```
(Iteration 3031 / 4900) loss: 1.369026
(Iteration 3041 / 4900) loss: 1.214198
(Iteration 3051 / 4900) loss: 1.143438
(Iteration 3061 / 4900) loss: 1.279604
(Iteration 3071 / 4900) loss: 1.373380
(Iteration 3081 / 4900) loss: 1.066720
(Iteration 3091 / 4900) loss: 1.069767
(Iteration 3101 / 4900) loss: 1.065370
(Iteration 3111 / 4900) loss: 1.361859
(Iteration 3121 / 4900) loss: 1.361362
(Iteration 3131 / 4900) loss: 1.014606
(Iteration 3141 / 4900) loss: 1.306831
(Iteration 3151 / 4900) loss: 1.333016
(Iteration 3161 / 4900) loss: 1.223335
(Iteration 3171 / 4900) loss: 1.264855
(Iteration 3181 / 4900) loss: 1.218897
(Iteration 3191 / 4900) loss: 1.249614
(Iteration 3201 / 4900) loss: 1.121298
(Iteration 3211 / 4900) loss: 1.267325
(Iteration 3221 / 4900) loss: 1.246610
(Iteration 3231 / 4900) loss: 1.091104
(Iteration 3241 / 4900) loss: 1.146100
(Iteration 3251 / 4900) loss: 0.998172
(Iteration 3261 / 4900) loss: 1.042157
(Iteration 3271 / 4900) loss: 1.583590
(Iteration 3281 / 4900) loss: 1.194515
(Iteration 3291 / 4900) loss: 1.253656
(Iteration 3301 / 4900) loss: 1.048516
(Iteration 3311 / 4900) loss: 1.274055
(Iteration 3321 / 4900) loss: 1.135561
(Iteration 3331 / 4900) loss: 1.122452
(Iteration 3341 / 4900) loss: 1.213772
(Iteration 3351 / 4900) loss: 1.235952
(Iteration 3361 / 4900) loss: 1.309886
(Iteration 3371 / 4900) loss: 1.433027
(Iteration 3381 / 4900) loss: 1.220397
(Iteration 3391 / 4900) loss: 1.206860
(Iteration 3401 / 4900) loss: 1.074450
(Iteration 3411 / 4900) loss: 1.416162
(Iteration 3421 / 4900) loss: 1.146718
(Epoch 7 / 10) train acc: 0.590000; val_acc: 0.470000
(Iteration 3431 / 4900) loss: 1.242975
(Iteration 3441 / 4900) loss: 1.139088
(Iteration 3451 / 4900) loss: 0.949546
(Iteration 3461 / 4900) loss: 1.308285
(Iteration 3471 / 4900) loss: 1.221694
(Iteration 3481 / 4900) loss: 1.057500
(Iteration 3491 / 4900) loss: 1.252922
```

```
(Iteration 3501 / 4900) loss: 1.247035
(Iteration 3511 / 4900) loss: 1.140896
(Iteration 3521 / 4900) loss: 1.029164
(Iteration 3531 / 4900) loss: 1.341288
(Iteration 3541 / 4900) loss: 1.064641
(Iteration 3551 / 4900) loss: 1.211427
(Iteration 3561 / 4900) loss: 1.137646
(Iteration 3571 / 4900) loss: 1.146061
(Iteration 3581 / 4900) loss: 1.124575
(Iteration 3591 / 4900) loss: 1.107300
(Iteration 3601 / 4900) loss: 1.068998
(Iteration 3611 / 4900) loss: 1.308220
(Iteration 3621 / 4900) loss: 1.257201
(Iteration 3631 / 4900) loss: 1.083568
(Iteration 3641 / 4900) loss: 1.429901
(Iteration 3651 / 4900) loss: 1.227528
(Iteration 3661 / 4900) loss: 1.232432
(Iteration 3671 / 4900) loss: 1.169048
(Iteration 3681 / 4900) loss: 1.270195
(Iteration 3691 / 4900) loss: 1.342897
(Iteration 3701 / 4900) loss: 1.239448
(Iteration 3711 / 4900) loss: 1.224940
(Iteration 3721 / 4900) loss: 1.098914
(Iteration 3731 / 4900) loss: 0.868718
(Iteration 3741 / 4900) loss: 1.104152
(Iteration 3751 / 4900) loss: 1.197668
(Iteration 3761 / 4900) loss: 1.132197
(Iteration 3771 / 4900) loss: 1.036901
(Iteration 3781 / 4900) loss: 1.252382
(Iteration 3791 / 4900) loss: 1.230459
(Iteration 3801 / 4900) loss: 1.195262
(Iteration 3811 / 4900) loss: 1.169530
(Iteration 3821 / 4900) loss: 1.218814
(Iteration 3831 / 4900) loss: 1.293966
(Iteration 3841 / 4900) loss: 1.209963
(Iteration 3851 / 4900) loss: 1.266488
(Iteration 3861 / 4900) loss: 1.189603
(Iteration 3871 / 4900) loss: 1.098935
(Iteration 3881 / 4900) loss: 1.043477
(Iteration 3891 / 4900) loss: 0.980716
(Iteration 3901 / 4900) loss: 1.161182
(Iteration 3911 / 4900) loss: 1.145245
(Epoch 8 / 10) train acc: 0.579000; val_acc: 0.523000
(Iteration 3921 / 4900) loss: 1.119351
(Iteration 3931 / 4900) loss: 1.301387
(Iteration 3941 / 4900) loss: 1.256559
(Iteration 3951 / 4900) loss: 1.237844
(Iteration 3961 / 4900) loss: 1.196610
```

```
(Iteration 3971 / 4900) loss: 1.021895
(Iteration 3981 / 4900) loss: 1.060830
(Iteration 3991 / 4900) loss: 1.090551
(Iteration 4001 / 4900) loss: 1.336074
(Iteration 4011 / 4900) loss: 1.293306
(Iteration 4021 / 4900) loss: 1.269853
(Iteration 4031 / 4900) loss: 1.301861
(Iteration 4041 / 4900) loss: 1.188977
(Iteration 4051 / 4900) loss: 1.152463
(Iteration 4061 / 4900) loss: 1.148890
(Iteration 4071 / 4900) loss: 1.145563
(Iteration 4081 / 4900) loss: 1.168948
(Iteration 4091 / 4900) loss: 1.419183
(Iteration 4101 / 4900) loss: 1.220704
(Iteration 4111 / 4900) loss: 1.022964
(Iteration 4121 / 4900) loss: 1.328968
(Iteration 4131 / 4900) loss: 1.437095
(Iteration 4141 / 4900) loss: 0.992239
(Iteration 4151 / 4900) loss: 1.114879
(Iteration 4161 / 4900) loss: 1.146670
(Iteration 4171 / 4900) loss: 0.924436
(Iteration 4181 / 4900) loss: 1.009740
(Iteration 4191 / 4900) loss: 1.123678
(Iteration 4201 / 4900) loss: 1.136269
(Iteration 4211 / 4900) loss: 1.200837
(Iteration 4221 / 4900) loss: 1.153187
(Iteration 4231 / 4900) loss: 1.102884
(Iteration 4241 / 4900) loss: 1.184444
(Iteration 4251 / 4900) loss: 1.095339
(Iteration 4261 / 4900) loss: 0.932696
(Iteration 4271 / 4900) loss: 1.039720
(Iteration 4281 / 4900) loss: 1.238935
(Iteration 4291 / 4900) loss: 1.273508
(Iteration 4301 / 4900) loss: 1.215600
(Iteration 4311 / 4900) loss: 1.113983
(Iteration 4321 / 4900) loss: 1.246508
(Iteration 4331 / 4900) loss: 1.036924
(Iteration 4341 / 4900) loss: 1.105446
(Iteration 4351 / 4900) loss: 1.143540
(Iteration 4361 / 4900) loss: 1.063420
(Iteration 4371 / 4900) loss: 1.153002
(Iteration 4381 / 4900) loss: 1.074084
(Iteration 4391 / 4900) loss: 1.074002
(Iteration 4401 / 4900) loss: 1.060275
(Epoch 9 / 10) train acc: 0.596000; val_acc: 0.501000
(Iteration 4411 / 4900) loss: 0.904393
(Iteration 4421 / 4900) loss: 1.058140
(Iteration 4431 / 4900) loss: 1.221801
```

```
(Iteration 4441 / 4900) loss: 1.203131
(Iteration 4451 / 4900) loss: 1.132795
(Iteration 4461 / 4900) loss: 1.289930
(Iteration 4471 / 4900) loss: 1.104910
(Iteration 4481 / 4900) loss: 1.030488
(Iteration 4491 / 4900) loss: 1.005103
(Iteration 4501 / 4900) loss: 1.148391
(Iteration 4511 / 4900) loss: 1.111396
(Iteration 4521 / 4900) loss: 1.172240
(Iteration 4531 / 4900) loss: 1.334594
(Iteration 4541 / 4900) loss: 1.045275
(Iteration 4551 / 4900) loss: 1.265855
(Iteration 4561 / 4900) loss: 1.253447
(Iteration 4571 / 4900) loss: 1.193882
(Iteration 4581 / 4900) loss: 1.091480
(Iteration 4591 / 4900) loss: 1.406596
(Iteration 4601 / 4900) loss: 1.349373
(Iteration 4611 / 4900) loss: 0.974045
(Iteration 4621 / 4900) loss: 1.252778
(Iteration 4631 / 4900) loss: 1.123981
(Iteration 4641 / 4900) loss: 1.092822
(Iteration 4651 / 4900) loss: 1.453715
(Iteration 4661 / 4900) loss: 1.151728
(Iteration 4671 / 4900) loss: 1.095514
(Iteration 4681 / 4900) loss: 1.059237
(Iteration 4691 / 4900) loss: 0.985982
(Iteration 4701 / 4900) loss: 1.043129
(Iteration 4711 / 4900) loss: 1.047601
(Iteration 4721 / 4900) loss: 1.220153
(Iteration 4731 / 4900) loss: 1.005459
(Iteration 4741 / 4900) loss: 1.005833
(Iteration 4751 / 4900) loss: 1.177115
(Iteration 4761 / 4900) loss: 1.156361
(Iteration 4771 / 4900) loss: 1.278099
(Iteration 4781 / 4900) loss: 1.078144
(Iteration 4791 / 4900) loss: 1.247884
(Iteration 4801 / 4900) loss: 1.004793
(Iteration 4811 / 4900) loss: 1.114070
(Iteration 4821 / 4900) loss: 1.023736
(Iteration 4831 / 4900) loss: 1.133469
(Iteration 4841 / 4900) loss: 1.133023
(Iteration 4851 / 4900) loss: 1.025839
(Iteration 4861 / 4900) loss: 1.234835
(Iteration 4871 / 4900) loss: 1.064527
(Iteration 4881 / 4900) loss: 1.012651
(Iteration 4891 / 4900) loss: 1.111463
(Epoch 10 / 10) train acc: 0.583000; val_acc: 0.490000
(Iteration 1 / 4900) loss: 2.302593
```

```
(Epoch 0 / 10) train acc: 0.099000; val_acc: 0.105000
(Iteration 11 / 4900) loss: 2.282046
(Iteration 21 / 4900) loss: 2.186514
(Iteration 31 / 4900) loss: 2.091416
(Iteration 41 / 4900) loss: 2.111068
(Iteration 51 / 4900) loss: 2.124002
(Iteration 61 / 4900) loss: 1.987547
(Iteration 71 / 4900) loss: 2.025512
(Iteration 81 / 4900) loss: 2.054207
(Iteration 91 / 4900) loss: 1.892680
(Iteration 101 / 4900) loss: 1.980422
(Iteration 111 / 4900) loss: 1.998791
(Iteration 121 / 4900) loss: 1.933749
(Iteration 131 / 4900) loss: 1.886809
(Iteration 141 / 4900) loss: 1.858444
(Iteration 151 / 4900) loss: 1.868329
(Iteration 161 / 4900) loss: 2.010396
(Iteration 171 / 4900) loss: 1.907972
(Iteration 181 / 4900) loss: 1.898358
(Iteration 191 / 4900) loss: 2.036476
(Iteration 201 / 4900) loss: 1.875856
(Iteration 211 / 4900) loss: 2.119043
(Iteration 221 / 4900) loss: 1.693402
(Iteration 231 / 4900) loss: 1.839682
(Iteration 241 / 4900) loss: 1.741625
(Iteration 251 / 4900) loss: 1.882384
(Iteration 261 / 4900) loss: 1.884584
(Iteration 271 / 4900) loss: 1.741885
(Iteration 281 / 4900) loss: 1.813728
(Iteration 291 / 4900) loss: 2.023645
(Iteration 301 / 4900) loss: 1.757699
(Iteration 311 / 4900) loss: 1.771704
(Iteration 321 / 4900) loss: 1.785252
(Iteration 331 / 4900) loss: 1.899646
(Iteration 341 / 4900) loss: 1.771111
(Iteration 351 / 4900) loss: 1.882417
(Iteration 361 / 4900) loss: 1.858106
(Iteration 371 / 4900) loss: 1.765302
(Iteration 381 / 4900) loss: 1.831240
(Iteration 391 / 4900) loss: 1.728114
(Iteration 401 / 4900) loss: 1.884276
(Iteration 411 / 4900) loss: 1.742589
(Iteration 421 / 4900) loss: 1.823882
(Iteration 431 / 4900) loss: 1.742531
(Iteration 441 / 4900) loss: 1.568696
(Iteration 451 / 4900) loss: 1.684933
(Iteration 461 / 4900) loss: 1.751843
(Iteration 471 / 4900) loss: 1.678443
```

```
(Iteration 481 / 4900) loss: 1.871464
(Epoch 1 / 10) train acc: 0.327000; val_acc: 0.335000
(Iteration 491 / 4900) loss: 1.698325
(Iteration 501 / 4900) loss: 1.784276
(Iteration 511 / 4900) loss: 1.734220
(Iteration 521 / 4900) loss: 1.868571
(Iteration 531 / 4900) loss: 1.606864
(Iteration 541 / 4900) loss: 1.638648
(Iteration 551 / 4900) loss: 1.576460
(Iteration 561 / 4900) loss: 2.013400
(Iteration 571 / 4900) loss: 1.831345
(Iteration 581 / 4900) loss: 1.518579
(Iteration 591 / 4900) loss: 1.680203
(Iteration 601 / 4900) loss: 1.847584
(Iteration 611 / 4900) loss: 1.594151
(Iteration 621 / 4900) loss: 1.722345
(Iteration 631 / 4900) loss: 1.608832
(Iteration 641 / 4900) loss: 1.576942
(Iteration 651 / 4900) loss: 1.527857
(Iteration 661 / 4900) loss: 1.767032
(Iteration 671 / 4900) loss: 1.625646
(Iteration 681 / 4900) loss: 1.829373
(Iteration 691 / 4900) loss: 1.811560
(Iteration 701 / 4900) loss: 1.537159
(Iteration 711 / 4900) loss: 1.543219
(Iteration 721 / 4900) loss: 1.599723
(Iteration 731 / 4900) loss: 1.652881
(Iteration 741 / 4900) loss: 1.622635
(Iteration 751 / 4900) loss: 1.469529
(Iteration 761 / 4900) loss: 1.697940
(Iteration 771 / 4900) loss: 1.905715
(Iteration 781 / 4900) loss: 1.523016
(Iteration 791 / 4900) loss: 1.542947
(Iteration 801 / 4900) loss: 1.517377
(Iteration 811 / 4900) loss: 1.718054
(Iteration 821 / 4900) loss: 1.674583
(Iteration 831 / 4900) loss: 1.816091
(Iteration 841 / 4900) loss: 1.507530
(Iteration 851 / 4900) loss: 1.580461
(Iteration 861 / 4900) loss: 1.778634
(Iteration 871 / 4900) loss: 1.746048
(Iteration 881 / 4900) loss: 1.726467
(Iteration 891 / 4900) loss: 1.658570
(Iteration 901 / 4900) loss: 1.465871
(Iteration 911 / 4900) loss: 1.690427
(Iteration 921 / 4900) loss: 1.533442
(Iteration 931 / 4900) loss: 1.588868
(Iteration 941 / 4900) loss: 1.603315
```

```
(Iteration 951 / 4900) loss: 1.530999
(Iteration 961 / 4900) loss: 1.609183
(Iteration 971 / 4900) loss: 1.521581
(Epoch 2 / 10) train acc: 0.437000; val_acc: 0.428000
(Iteration 981 / 4900) loss: 1.668688
(Iteration 991 / 4900) loss: 1.439597
(Iteration 1001 / 4900) loss: 1.492005
(Iteration 1011 / 4900) loss: 1.503218
(Iteration 1021 / 4900) loss: 1.513400
(Iteration 1031 / 4900) loss: 1.549303
(Iteration 1041 / 4900) loss: 1.567522
(Iteration 1051 / 4900) loss: 1.405992
(Iteration 1061 / 4900) loss: 1.398755
(Iteration 1071 / 4900) loss: 1.695594
(Iteration 1081 / 4900) loss: 1.528424
(Iteration 1091 / 4900) loss: 1.503610
(Iteration 1101 / 4900) loss: 1.455840
(Iteration 1111 / 4900) loss: 1.572172
(Iteration 1121 / 4900) loss: 1.421039
(Iteration 1131 / 4900) loss: 1.481450
(Iteration 1141 / 4900) loss: 1.325852
(Iteration 1151 / 4900) loss: 1.425812
(Iteration 1161 / 4900) loss: 1.423976
(Iteration 1171 / 4900) loss: 1.354732
(Iteration 1181 / 4900) loss: 1.600923
(Iteration 1191 / 4900) loss: 1.435034
(Iteration 1201 / 4900) loss: 1.515864
(Iteration 1211 / 4900) loss: 1.606558
(Iteration 1221 / 4900) loss: 1.811161
(Iteration 1231 / 4900) loss: 1.407660
(Iteration 1241 / 4900) loss: 1.524112
(Iteration 1251 / 4900) loss: 1.436403
(Iteration 1261 / 4900) loss: 1.583015
(Iteration 1271 / 4900) loss: 1.525369
(Iteration 1281 / 4900) loss: 1.309499
(Iteration 1291 / 4900) loss: 1.422473
(Iteration 1301 / 4900) loss: 1.601840
(Iteration 1311 / 4900) loss: 1.484756
(Iteration 1321 / 4900) loss: 1.506004
(Iteration 1331 / 4900) loss: 1.353281
(Iteration 1341 / 4900) loss: 1.508600
(Iteration 1351 / 4900) loss: 1.410488
(Iteration 1361 / 4900) loss: 1.385721
(Iteration 1371 / 4900) loss: 1.447355
(Iteration 1381 / 4900) loss: 1.675235
(Iteration 1391 / 4900) loss: 1.454253
(Iteration 1401 / 4900) loss: 1.458186
(Iteration 1411 / 4900) loss: 1.376390
```

```
(Iteration 1421 / 4900) loss: 1.398655
(Iteration 1431 / 4900) loss: 1.289132
(Iteration 1441 / 4900) loss: 1.473320
(Iteration 1451 / 4900) loss: 1.427418
(Iteration 1461 / 4900) loss: 1.436839
(Epoch 3 / 10) train acc: 0.494000; val_acc: 0.463000
(Iteration 1471 / 4900) loss: 1.408784
(Iteration 1481 / 4900) loss: 1.419566
(Iteration 1491 / 4900) loss: 1.435206
(Iteration 1501 / 4900) loss: 1.588022
(Iteration 1511 / 4900) loss: 1.330016
(Iteration 1521 / 4900) loss: 1.577846
(Iteration 1531 / 4900) loss: 1.440481
(Iteration 1541 / 4900) loss: 1.419137
(Iteration 1551 / 4900) loss: 1.445982
(Iteration 1561 / 4900) loss: 1.344281
(Iteration 1571 / 4900) loss: 1.621841
(Iteration 1581 / 4900) loss: 1.323162
(Iteration 1591 / 4900) loss: 1.544906
(Iteration 1601 / 4900) loss: 1.442957
(Iteration 1611 / 4900) loss: 1.393896
(Iteration 1621 / 4900) loss: 1.404909
(Iteration 1631 / 4900) loss: 1.354444
(Iteration 1641 / 4900) loss: 1.384889
(Iteration 1651 / 4900) loss: 1.394653
(Iteration 1661 / 4900) loss: 1.501142
(Iteration 1671 / 4900) loss: 1.298425
(Iteration 1681 / 4900) loss: 1.388792
(Iteration 1691 / 4900) loss: 1.384974
(Iteration 1701 / 4900) loss: 1.358982
(Iteration 1711 / 4900) loss: 1.352829
(Iteration 1721 / 4900) loss: 1.401005
(Iteration 1731 / 4900) loss: 1.317041
(Iteration 1741 / 4900) loss: 1.364062
(Iteration 1751 / 4900) loss: 1.386858
(Iteration 1761 / 4900) loss: 1.388080
(Iteration 1771 / 4900) loss: 1.330811
(Iteration 1781 / 4900) loss: 1.257375
(Iteration 1791 / 4900) loss: 1.463578
(Iteration 1801 / 4900) loss: 1.299576
(Iteration 1811 / 4900) loss: 1.311235
(Iteration 1821 / 4900) loss: 1.399483
(Iteration 1831 / 4900) loss: 1.336979
(Iteration 1841 / 4900) loss: 1.502635
(Iteration 1851 / 4900) loss: 1.343965
(Iteration 1861 / 4900) loss: 1.275904
(Iteration 1871 / 4900) loss: 1.554422
(Iteration 1881 / 4900) loss: 1.565505
```

```
(Iteration 1891 / 4900) loss: 1.241157
(Iteration 1901 / 4900) loss: 1.404786
(Iteration 1911 / 4900) loss: 1.390664
(Iteration 1921 / 4900) loss: 1.457971
(Iteration 1931 / 4900) loss: 1.357851
(Iteration 1941 / 4900) loss: 1.345889
(Iteration 1951 / 4900) loss: 1.184887
(Epoch 4 / 10) train acc: 0.497000; val_acc: 0.497000
(Iteration 1961 / 4900) loss: 1.395044
(Iteration 1971 / 4900) loss: 1.479751
(Iteration 1981 / 4900) loss: 1.314307
(Iteration 1991 / 4900) loss: 1.505485
(Iteration 2001 / 4900) loss: 1.295735
(Iteration 2011 / 4900) loss: 1.496515
(Iteration 2021 / 4900) loss: 1.359155
(Iteration 2031 / 4900) loss: 1.389087
(Iteration 2041 / 4900) loss: 1.429294
(Iteration 2051 / 4900) loss: 1.333859
(Iteration 2061 / 4900) loss: 1.343915
(Iteration 2071 / 4900) loss: 1.504596
(Iteration 2081 / 4900) loss: 1.440437
(Iteration 2091 / 4900) loss: 1.325402
(Iteration 2101 / 4900) loss: 1.457258
(Iteration 2111 / 4900) loss: 1.340778
(Iteration 2121 / 4900) loss: 1.079138
(Iteration 2131 / 4900) loss: 1.317477
(Iteration 2141 / 4900) loss: 1.249804
(Iteration 2151 / 4900) loss: 1.424285
(Iteration 2161 / 4900) loss: 1.178856
(Iteration 2171 / 4900) loss: 1.243257
(Iteration 2181 / 4900) loss: 1.374982
(Iteration 2191 / 4900) loss: 1.210496
(Iteration 2201 / 4900) loss: 1.249560
(Iteration 2211 / 4900) loss: 1.314944
(Iteration 2221 / 4900) loss: 1.308445
(Iteration 2231 / 4900) loss: 1.344910
(Iteration 2241 / 4900) loss: 1.191521
(Iteration 2251 / 4900) loss: 1.200567
(Iteration 2261 / 4900) loss: 1.434623
(Iteration 2271 / 4900) loss: 1.320315
(Iteration 2281 / 4900) loss: 1.335023
(Iteration 2291 / 4900) loss: 1.435176
(Iteration 2301 / 4900) loss: 1.344342
(Iteration 2311 / 4900) loss: 1.202608
(Iteration 2321 / 4900) loss: 1.266456
(Iteration 2331 / 4900) loss: 1.234887
(Iteration 2341 / 4900) loss: 1.432199
(Iteration 2351 / 4900) loss: 1.201944
```

```
(Iteration 2361 / 4900) loss: 1.290466
(Iteration 2371 / 4900) loss: 1.457767
(Iteration 2381 / 4900) loss: 1.305220
(Iteration 2391 / 4900) loss: 1.145657
(Iteration 2401 / 4900) loss: 1.220728
(Iteration 2411 / 4900) loss: 1.290210
(Iteration 2421 / 4900) loss: 1.315576
(Iteration 2431 / 4900) loss: 1.087134
(Iteration 2441 / 4900) loss: 1.500019
(Epoch 5 / 10) train acc: 0.554000; val_acc: 0.492000
(Iteration 2451 / 4900) loss: 1.228915
(Iteration 2461 / 4900) loss: 1.344671
(Iteration 2471 / 4900) loss: 1.264783
(Iteration 2481 / 4900) loss: 1.311846
(Iteration 2491 / 4900) loss: 1.236665
(Iteration 2501 / 4900) loss: 1.332385
(Iteration 2511 / 4900) loss: 1.305344
(Iteration 2521 / 4900) loss: 1.404343
(Iteration 2531 / 4900) loss: 1.231079
(Iteration 2541 / 4900) loss: 1.507208
(Iteration 2551 / 4900) loss: 1.557599
(Iteration 2561 / 4900) loss: 1.275892
(Iteration 2571 / 4900) loss: 1.269429
(Iteration 2581 / 4900) loss: 1.081782
(Iteration 2591 / 4900) loss: 1.308510
(Iteration 2601 / 4900) loss: 1.362001
(Iteration 2611 / 4900) loss: 1.310999
(Iteration 2621 / 4900) loss: 1.084180
(Iteration 2631 / 4900) loss: 1.414872
(Iteration 2641 / 4900) loss: 1.465317
(Iteration 2651 / 4900) loss: 1.183704
(Iteration 2661 / 4900) loss: 1.307656
(Iteration 2671 / 4900) loss: 1.198114
(Iteration 2681 / 4900) loss: 1.369741
(Iteration 2691 / 4900) loss: 1.089967
(Iteration 2701 / 4900) loss: 1.347249
(Iteration 2711 / 4900) loss: 1.245108
(Iteration 2721 / 4900) loss: 1.444092
(Iteration 2731 / 4900) loss: 1.301923
(Iteration 2741 / 4900) loss: 1.347101
(Iteration 2751 / 4900) loss: 1.294298
(Iteration 2761 / 4900) loss: 1.321939
(Iteration 2771 / 4900) loss: 1.268454
(Iteration 2781 / 4900) loss: 1.295857
(Iteration 2791 / 4900) loss: 1.209130
(Iteration 2801 / 4900) loss: 1.382088
(Iteration 2811 / 4900) loss: 1.337837
(Iteration 2821 / 4900) loss: 1.025677
```

```
(Iteration 2831 / 4900) loss: 1.289669
(Iteration 2841 / 4900) loss: 1.108295
(Iteration 2851 / 4900) loss: 1.189489
(Iteration 2861 / 4900) loss: 1.403086
(Iteration 2871 / 4900) loss: 1.318090
(Iteration 2881 / 4900) loss: 1.432886
(Iteration 2891 / 4900) loss: 1.369666
(Iteration 2901 / 4900) loss: 1.364884
(Iteration 2911 / 4900) loss: 1.128317
(Iteration 2921 / 4900) loss: 1.401802
(Iteration 2931 / 4900) loss: 1.173870
(Epoch 6 / 10) train acc: 0.571000; val_acc: 0.511000
(Iteration 2941 / 4900) loss: 1.185254
(Iteration 2951 / 4900) loss: 1.104686
(Iteration 2961 / 4900) loss: 1.246556
(Iteration 2971 / 4900) loss: 1.231562
(Iteration 2981 / 4900) loss: 1.247088
(Iteration 2991 / 4900) loss: 1.050204
(Iteration 3001 / 4900) loss: 1.351485
(Iteration 3011 / 4900) loss: 1.426476
(Iteration 3021 / 4900) loss: 0.896249
(Iteration 3031 / 4900) loss: 1.129711
(Iteration 3041 / 4900) loss: 1.395729
(Iteration 3051 / 4900) loss: 1.371017
(Iteration 3061 / 4900) loss: 1.511436
(Iteration 3071 / 4900) loss: 1.166102
(Iteration 3081 / 4900) loss: 1.328965
(Iteration 3091 / 4900) loss: 1.188886
(Iteration 3101 / 4900) loss: 1.305870
(Iteration 3111 / 4900) loss: 1.143152
(Iteration 3121 / 4900) loss: 1.198552
(Iteration 3131 / 4900) loss: 1.226617
(Iteration 3141 / 4900) loss: 1.219047
(Iteration 3151 / 4900) loss: 1.213003
(Iteration 3161 / 4900) loss: 1.272248
(Iteration 3171 / 4900) loss: 1.162494
(Iteration 3181 / 4900) loss: 1.212550
(Iteration 3191 / 4900) loss: 1.378385
(Iteration 3201 / 4900) loss: 1.359449
(Iteration 3211 / 4900) loss: 1.039940
(Iteration 3221 / 4900) loss: 1.303778
(Iteration 3231 / 4900) loss: 1.309921
(Iteration 3241 / 4900) loss: 1.351869
(Iteration 3251 / 4900) loss: 1.123516
(Iteration 3261 / 4900) loss: 1.242199
(Iteration 3271 / 4900) loss: 1.103289
(Iteration 3281 / 4900) loss: 1.222449
(Iteration 3291 / 4900) loss: 1.163005
```

```
(Iteration 3301 / 4900) loss: 1.415695
(Iteration 3311 / 4900) loss: 1.294606
(Iteration 3321 / 4900) loss: 1.233759
(Iteration 3331 / 4900) loss: 1.070782
(Iteration 3341 / 4900) loss: 1.195325
(Iteration 3351 / 4900) loss: 1.265374
(Iteration 3361 / 4900) loss: 1.170340
(Iteration 3371 / 4900) loss: 1.244825
(Iteration 3381 / 4900) loss: 1.103787
(Iteration 3391 / 4900) loss: 1.199510
(Iteration 3401 / 4900) loss: 1.274261
(Iteration 3411 / 4900) loss: 1.545436
(Iteration 3421 / 4900) loss: 1.274023
(Epoch 7 / 10) train acc: 0.539000; val_acc: 0.512000
(Iteration 3431 / 4900) loss: 0.940006
(Iteration 3441 / 4900) loss: 1.201469
(Iteration 3451 / 4900) loss: 1.006275
(Iteration 3461 / 4900) loss: 1.053289
(Iteration 3471 / 4900) loss: 1.293350
(Iteration 3481 / 4900) loss: 1.082243
(Iteration 3491 / 4900) loss: 1.252254
(Iteration 3501 / 4900) loss: 1.186253
(Iteration 3511 / 4900) loss: 1.172676
(Iteration 3521 / 4900) loss: 1.043405
(Iteration 3531 / 4900) loss: 1.070556
(Iteration 3541 / 4900) loss: 1.184053
(Iteration 3551 / 4900) loss: 1.204158
(Iteration 3561 / 4900) loss: 1.008537
(Iteration 3571 / 4900) loss: 1.174369
(Iteration 3581 / 4900) loss: 1.064480
(Iteration 3591 / 4900) loss: 1.161893
(Iteration 3601 / 4900) loss: 1.138164
(Iteration 3611 / 4900) loss: 1.007910
(Iteration 3621 / 4900) loss: 1.168809
(Iteration 3631 / 4900) loss: 1.162173
(Iteration 3641 / 4900) loss: 1.294855
(Iteration 3651 / 4900) loss: 1.146118
(Iteration 3661 / 4900) loss: 1.185306
(Iteration 3671 / 4900) loss: 1.012666
(Iteration 3681 / 4900) loss: 1.173326
(Iteration 3691 / 4900) loss: 1.419361
(Iteration 3701 / 4900) loss: 1.372065
(Iteration 3711 / 4900) loss: 1.320122
(Iteration 3721 / 4900) loss: 1.083910
(Iteration 3731 / 4900) loss: 1.103335
(Iteration 3741 / 4900) loss: 1.196586
(Iteration 3751 / 4900) loss: 1.189001
(Iteration 3761 / 4900) loss: 1.254293
```

```
(Iteration 3771 / 4900) loss: 1.161378
(Iteration 3781 / 4900) loss: 1.330193
(Iteration 3791 / 4900) loss: 1.052481
(Iteration 3801 / 4900) loss: 1.054283
(Iteration 3811 / 4900) loss: 1.262289
(Iteration 3821 / 4900) loss: 1.202271
(Iteration 3831 / 4900) loss: 1.253163
(Iteration 3841 / 4900) loss: 1.085177
(Iteration 3851 / 4900) loss: 1.142002
(Iteration 3861 / 4900) loss: 1.364094
(Iteration 3871 / 4900) loss: 1.135016
(Iteration 3881 / 4900) loss: 1.055091
(Iteration 3891 / 4900) loss: 1.081454
(Iteration 3901 / 4900) loss: 1.210458
(Iteration 3911 / 4900) loss: 1.017023
(Epoch 8 / 10) train acc: 0.582000; val_acc: 0.528000
(Iteration 3921 / 4900) loss: 1.252798
(Iteration 3931 / 4900) loss: 1.173106
(Iteration 3941 / 4900) loss: 1.104459
(Iteration 3951 / 4900) loss: 1.022904
(Iteration 3961 / 4900) loss: 0.985581
(Iteration 3971 / 4900) loss: 1.078840
(Iteration 3981 / 4900) loss: 1.324598
(Iteration 3991 / 4900) loss: 1.062297
(Iteration 4001 / 4900) loss: 1.141521
(Iteration 4011 / 4900) loss: 1.130754
(Iteration 4021 / 4900) loss: 1.170816
(Iteration 4031 / 4900) loss: 1.047193
(Iteration 4041 / 4900) loss: 1.135895
(Iteration 4051 / 4900) loss: 1.016157
(Iteration 4061 / 4900) loss: 1.145402
(Iteration 4071 / 4900) loss: 1.248418
(Iteration 4081 / 4900) loss: 1.101833
(Iteration 4091 / 4900) loss: 0.979608
(Iteration 4101 / 4900) loss: 1.233541
(Iteration 4111 / 4900) loss: 1.072136
(Iteration 4121 / 4900) loss: 1.155314
(Iteration 4131 / 4900) loss: 0.950807
(Iteration 4141 / 4900) loss: 1.213827
(Iteration 4151 / 4900) loss: 1.092753
(Iteration 4161 / 4900) loss: 1.055206
(Iteration 4171 / 4900) loss: 1.173171
(Iteration 4181 / 4900) loss: 1.023451
(Iteration 4191 / 4900) loss: 1.227422
(Iteration 4201 / 4900) loss: 1.106634
(Iteration 4211 / 4900) loss: 1.128865
(Iteration 4221 / 4900) loss: 1.160361
(Iteration 4231 / 4900) loss: 1.304307
```

```
(Iteration 4241 / 4900) loss: 0.839043
(Iteration 4251 / 4900) loss: 1.276764
(Iteration 4261 / 4900) loss: 1.220832
(Iteration 4271 / 4900) loss: 1.115908
(Iteration 4281 / 4900) loss: 1.139936
(Iteration 4291 / 4900) loss: 0.956679
(Iteration 4301 / 4900) loss: 1.116928
(Iteration 4311 / 4900) loss: 1.034071
(Iteration 4321 / 4900) loss: 1.119290
(Iteration 4331 / 4900) loss: 1.210407
(Iteration 4341 / 4900) loss: 1.301325
(Iteration 4351 / 4900) loss: 1.126358
(Iteration 4361 / 4900) loss: 1.027943
(Iteration 4371 / 4900) loss: 1.197311
(Iteration 4381 / 4900) loss: 1.023772
(Iteration 4391 / 4900) loss: 0.876461
(Iteration 4401 / 4900) loss: 0.790674
(Epoch 9 / 10) train acc: 0.603000; val_acc: 0.517000
(Iteration 4411 / 4900) loss: 1.056699
(Iteration 4421 / 4900) loss: 1.207837
(Iteration 4431 / 4900) loss: 1.198719
(Iteration 4441 / 4900) loss: 0.979982
(Iteration 4451 / 4900) loss: 0.973722
(Iteration 4461 / 4900) loss: 1.146732
(Iteration 4471 / 4900) loss: 1.183295
(Iteration 4481 / 4900) loss: 1.127616
(Iteration 4491 / 4900) loss: 1.078788
(Iteration 4501 / 4900) loss: 1.085796
(Iteration 4511 / 4900) loss: 1.149045
(Iteration 4521 / 4900) loss: 1.245687
(Iteration 4531 / 4900) loss: 1.176837
(Iteration 4541 / 4900) loss: 1.141317
(Iteration 4551 / 4900) loss: 1.195595
(Iteration 4561 / 4900) loss: 1.182826
(Iteration 4571 / 4900) loss: 1.139777
(Iteration 4581 / 4900) loss: 1.037792
(Iteration 4591 / 4900) loss: 1.191851
(Iteration 4601 / 4900) loss: 1.044332
(Iteration 4611 / 4900) loss: 1.299266
(Iteration 4621 / 4900) loss: 1.149527
(Iteration 4631 / 4900) loss: 1.150932
(Iteration 4641 / 4900) loss: 1.269417
(Iteration 4651 / 4900) loss: 1.111544
(Iteration 4661 / 4900) loss: 1.287309
(Iteration 4671 / 4900) loss: 1.085334
(Iteration 4681 / 4900) loss: 1.063827
(Iteration 4691 / 4900) loss: 0.991468
(Iteration 4701 / 4900) loss: 1.319235
```

```
(Iteration 4711 / 4900) loss: 1.217541
(Iteration 4721 / 4900) loss: 1.161966
(Iteration 4731 / 4900) loss: 0.942206
(Iteration 4741 / 4900) loss: 1.207049
(Iteration 4751 / 4900) loss: 1.032032
(Iteration 4761 / 4900) loss: 0.912067
(Iteration 4771 / 4900) loss: 0.924734
(Iteration 4781 / 4900) loss: 1.207087
(Iteration 4791 / 4900) loss: 1.045282
(Iteration 4801 / 4900) loss: 1.205337
(Iteration 4811 / 4900) loss: 1.114581
(Iteration 4821 / 4900) loss: 0.925237
(Iteration 4831 / 4900) loss: 1.164173
(Iteration 4841 / 4900) loss: 1.153418
(Iteration 4851 / 4900) loss: 1.066801
(Iteration 4861 / 4900) loss: 1.287543
(Iteration 4871 / 4900) loss: 1.145052
(Iteration 4881 / 4900) loss: 1.153079
(Iteration 4891 / 4900) loss: 1.224469
(Epoch 10 / 10) train acc: 0.609000; val_acc: 0.544000
(Iteration 1 / 4900) loss: 2.698706
(Epoch 0 / 10) train acc: 0.113000; val_acc: 0.111000
(Iteration 11 / 4900) loss: 2.235467
(Iteration 21 / 4900) loss: 1.997886
(Iteration 31 / 4900) loss: 2.067956
(Iteration 41 / 4900) loss: 1.951963
(Iteration 51 / 4900) loss: 1.801043
(Iteration 61 / 4900) loss: 1.853295
(Iteration 71 / 4900) loss: 1.807505
(Iteration 81 / 4900) loss: 1.829022
(Iteration 91 / 4900) loss: 1.847788
(Iteration 101 / 4900) loss: 1.834400
(Iteration 111 / 4900) loss: 1.756890
(Iteration 121 / 4900) loss: 1.797122
(Iteration 131 / 4900) loss: 1.898164
(Iteration 141 / 4900) loss: 1.704904
(Iteration 151 / 4900) loss: 1.731218
(Iteration 161 / 4900) loss: 1.944278
(Iteration 171 / 4900) loss: 1.815521
(Iteration 181 / 4900) loss: 1.767678
(Iteration 191 / 4900) loss: 1.631181
(Iteration 201 / 4900) loss: 1.631621
(Iteration 211 / 4900) loss: 1.754006
(Iteration 221 / 4900) loss: 1.998625
(Iteration 231 / 4900) loss: 1.648706
(Iteration 241 / 4900) loss: 1.528078
(Iteration 251 / 4900) loss: 1.755259
(Iteration 261 / 4900) loss: 1.741780
```

```
(Iteration 271 / 4900) loss: 1.503423
(Iteration 281 / 4900) loss: 1.476794
(Iteration 291 / 4900) loss: 1.784112
(Iteration 301 / 4900) loss: 1.596662
(Iteration 311 / 4900) loss: 1.391033
(Iteration 321 / 4900) loss: 1.576448
(Iteration 331 / 4900) loss: 1.648156
(Iteration 341 / 4900) loss: 1.484339
(Iteration 351 / 4900) loss: 1.695150
(Iteration 361 / 4900) loss: 1.624948
(Iteration 371 / 4900) loss: 1.402468
(Iteration 381 / 4900) loss: 1.403008
(Iteration 391 / 4900) loss: 1.679244
(Iteration 401 / 4900) loss: 1.720533
(Iteration 411 / 4900) loss: 1.734431
(Iteration 421 / 4900) loss: 1.505375
(Iteration 431 / 4900) loss: 1.686897
(Iteration 441 / 4900) loss: 1.429476
(Iteration 451 / 4900) loss: 1.575827
(Iteration 461 / 4900) loss: 1.629103
(Iteration 471 / 4900) loss: 1.572830
(Iteration 481 / 4900) loss: 1.559501
(Epoch 1 / 10) train acc: 0.440000; val_acc: 0.422000
(Iteration 491 / 4900) loss: 1.376283
(Iteration 501 / 4900) loss: 1.693669
(Iteration 511 / 4900) loss: 1.581000
(Iteration 521 / 4900) loss: 1.595558
(Iteration 531 / 4900) loss: 1.558769
(Iteration 541 / 4900) loss: 1.449497
(Iteration 551 / 4900) loss: 1.583242
(Iteration 561 / 4900) loss: 1.513461
(Iteration 571 / 4900) loss: 1.281630
(Iteration 581 / 4900) loss: 1.852362
(Iteration 591 / 4900) loss: 1.563734
(Iteration 601 / 4900) loss: 1.494286
(Iteration 611 / 4900) loss: 1.770226
(Iteration 621 / 4900) loss: 1.412646
(Iteration 631 / 4900) loss: 1.435524
(Iteration 641 / 4900) loss: 1.562244
(Iteration 651 / 4900) loss: 1.478936
(Iteration 661 / 4900) loss: 1.561431
(Iteration 671 / 4900) loss: 1.546828
(Iteration 681 / 4900) loss: 1.493262
(Iteration 691 / 4900) loss: 1.388133
(Iteration 701 / 4900) loss: 1.592300
(Iteration 711 / 4900) loss: 1.547956
(Iteration 721 / 4900) loss: 1.619091
(Iteration 731 / 4900) loss: 1.550138
```

```
(Iteration 741 / 4900) loss: 1.532184
(Iteration 751 / 4900) loss: 1.355102
(Iteration 761 / 4900) loss: 1.655549
(Iteration 771 / 4900) loss: 1.746888
(Iteration 781 / 4900) loss: 1.483095
(Iteration 791 / 4900) loss: 1.464119
(Iteration 801 / 4900) loss: 1.447234
(Iteration 811 / 4900) loss: 1.501475
(Iteration 821 / 4900) loss: 1.235544
(Iteration 831 / 4900) loss: 1.683094
(Iteration 841 / 4900) loss: 1.643483
(Iteration 851 / 4900) loss: 1.446975
(Iteration 861 / 4900) loss: 1.367609
(Iteration 871 / 4900) loss: 1.255659
(Iteration 881 / 4900) loss: 1.550824
(Iteration 891 / 4900) loss: 1.462284
(Iteration 901 / 4900) loss: 1.741480
(Iteration 911 / 4900) loss: 1.204656
(Iteration 921 / 4900) loss: 1.445016
(Iteration 931 / 4900) loss: 1.456727
(Iteration 941 / 4900) loss: 1.492145
(Iteration 951 / 4900) loss: 1.450300
(Iteration 961 / 4900) loss: 1.579407
(Iteration 971 / 4900) loss: 1.303393
(Epoch 2 / 10) train acc: 0.483000; val_acc: 0.462000
(Iteration 981 / 4900) loss: 1.410058
(Iteration 991 / 4900) loss: 1.555821
(Iteration 1001 / 4900) loss: 1.441343
(Iteration 1011 / 4900) loss: 1.345431
(Iteration 1021 / 4900) loss: 1.465871
(Iteration 1031 / 4900) loss: 1.489716
(Iteration 1041 / 4900) loss: 1.570248
(Iteration 1051 / 4900) loss: 1.286725
(Iteration 1061 / 4900) loss: 1.418924
(Iteration 1071 / 4900) loss: 1.356140
(Iteration 1081 / 4900) loss: 1.409848
(Iteration 1091 / 4900) loss: 1.234150
(Iteration 1101 / 4900) loss: 1.551546
(Iteration 1111 / 4900) loss: 1.301065
(Iteration 1121 / 4900) loss: 1.471840
(Iteration 1131 / 4900) loss: 1.426111
(Iteration 1141 / 4900) loss: 1.524584
(Iteration 1151 / 4900) loss: 1.356827
(Iteration 1161 / 4900) loss: 1.509683
(Iteration 1171 / 4900) loss: 1.571576
(Iteration 1181 / 4900) loss: 1.290543
(Iteration 1191 / 4900) loss: 1.495986
(Iteration 1201 / 4900) loss: 1.564570
```

```
(Iteration 1211 / 4900) loss: 1.325141
(Iteration 1221 / 4900) loss: 1.421811
(Iteration 1231 / 4900) loss: 1.549318
(Iteration 1241 / 4900) loss: 1.296744
(Iteration 1251 / 4900) loss: 1.228405
(Iteration 1261 / 4900) loss: 1.226908
(Iteration 1271 / 4900) loss: 1.385556
(Iteration 1281 / 4900) loss: 1.157446
(Iteration 1291 / 4900) loss: 1.492776
(Iteration 1301 / 4900) loss: 1.180744
(Iteration 1311 / 4900) loss: 1.378371
(Iteration 1321 / 4900) loss: 1.503553
(Iteration 1331 / 4900) loss: 1.144731
(Iteration 1341 / 4900) loss: 1.335813
(Iteration 1351 / 4900) loss: 1.551235
(Iteration 1361 / 4900) loss: 1.561759
(Iteration 1371 / 4900) loss: 1.593350
(Iteration 1381 / 4900) loss: 1.438762
(Iteration 1391 / 4900) loss: 1.251150
(Iteration 1401 / 4900) loss: 1.181484
(Iteration 1411 / 4900) loss: 1.460838
(Iteration 1421 / 4900) loss: 1.333609
(Iteration 1431 / 4900) loss: 1.354261
(Iteration 1441 / 4900) loss: 1.191813
(Iteration 1451 / 4900) loss: 1.544205
(Iteration 1461 / 4900) loss: 1.418606
(Epoch 3 / 10) train acc: 0.529000; val_acc: 0.469000
(Iteration 1471 / 4900) loss: 1.259965
(Iteration 1481 / 4900) loss: 1.348785
(Iteration 1491 / 4900) loss: 1.523940
(Iteration 1501 / 4900) loss: 1.337736
(Iteration 1511 / 4900) loss: 1.305560
(Iteration 1521 / 4900) loss: 1.444030
(Iteration 1531 / 4900) loss: 1.430221
(Iteration 1541 / 4900) loss: 1.292094
(Iteration 1551 / 4900) loss: 1.245455
(Iteration 1561 / 4900) loss: 1.259117
(Iteration 1571 / 4900) loss: 1.123142
(Iteration 1581 / 4900) loss: 1.317415
(Iteration 1591 / 4900) loss: 1.228215
(Iteration 1601 / 4900) loss: 1.257016
(Iteration 1611 / 4900) loss: 1.319864
(Iteration 1621 / 4900) loss: 1.444217
(Iteration 1631 / 4900) loss: 1.328009
(Iteration 1641 / 4900) loss: 1.337854
(Iteration 1651 / 4900) loss: 1.280647
(Iteration 1661 / 4900) loss: 1.289487
(Iteration 1671 / 4900) loss: 1.366259
```

```
(Iteration 1681 / 4900) loss: 1.496638
(Iteration 1691 / 4900) loss: 1.254363
(Iteration 1701 / 4900) loss: 1.216099
(Iteration 1711 / 4900) loss: 1.417824
(Iteration 1721 / 4900) loss: 1.304428
(Iteration 1731 / 4900) loss: 1.207976
(Iteration 1741 / 4900) loss: 1.287895
(Iteration 1751 / 4900) loss: 1.192035
(Iteration 1761 / 4900) loss: 1.274234
(Iteration 1771 / 4900) loss: 1.298526
(Iteration 1781 / 4900) loss: 1.326680
(Iteration 1791 / 4900) loss: 1.319239
(Iteration 1801 / 4900) loss: 1.384295
(Iteration 1811 / 4900) loss: 1.235736
(Iteration 1821 / 4900) loss: 1.201484
(Iteration 1831 / 4900) loss: 1.371474
(Iteration 1841 / 4900) loss: 1.209262
(Iteration 1851 / 4900) loss: 1.374319
(Iteration 1861 / 4900) loss: 1.255956
(Iteration 1871 / 4900) loss: 1.289207
(Iteration 1881 / 4900) loss: 1.308800
(Iteration 1891 / 4900) loss: 1.158420
(Iteration 1901 / 4900) loss: 1.137168
(Iteration 1911 / 4900) loss: 1.505674
(Iteration 1921 / 4900) loss: 1.325331
(Iteration 1931 / 4900) loss: 1.378380
(Iteration 1941 / 4900) loss: 1.179362
(Iteration 1951 / 4900) loss: 1.363818
(Epoch 4 / 10) train acc: 0.555000; val_acc: 0.467000
(Iteration 1961 / 4900) loss: 1.450206
(Iteration 1971 / 4900) loss: 1.117826
(Iteration 1981 / 4900) loss: 1.326115
(Iteration 1991 / 4900) loss: 1.179286
(Iteration 2001 / 4900) loss: 1.197303
(Iteration 2011 / 4900) loss: 1.531199
(Iteration 2021 / 4900) loss: 1.253674
(Iteration 2031 / 4900) loss: 1.085810
(Iteration 2041 / 4900) loss: 1.214490
(Iteration 2051 / 4900) loss: 1.234205
(Iteration 2061 / 4900) loss: 1.363442
(Iteration 2071 / 4900) loss: 1.187128
(Iteration 2081 / 4900) loss: 1.138143
(Iteration 2091 / 4900) loss: 1.319179
(Iteration 2101 / 4900) loss: 1.400723
(Iteration 2111 / 4900) loss: 1.130135
(Iteration 2121 / 4900) loss: 1.303841
(Iteration 2131 / 4900) loss: 1.323589
(Iteration 2141 / 4900) loss: 1.197679
```

```
(Iteration 2151 / 4900) loss: 1.469385
(Iteration 2161 / 4900) loss: 1.161271
(Iteration 2171 / 4900) loss: 1.243614
(Iteration 2181 / 4900) loss: 1.062688
(Iteration 2191 / 4900) loss: 1.246155
(Iteration 2201 / 4900) loss: 1.206994
(Iteration 2211 / 4900) loss: 1.146767
(Iteration 2221 / 4900) loss: 1.254660
(Iteration 2231 / 4900) loss: 1.278506
(Iteration 2241 / 4900) loss: 1.288378
(Iteration 2251 / 4900) loss: 1.268151
(Iteration 2261 / 4900) loss: 1.255821
(Iteration 2271 / 4900) loss: 1.173908
(Iteration 2281 / 4900) loss: 1.224221
(Iteration 2291 / 4900) loss: 1.301675
(Iteration 2301 / 4900) loss: 1.252085
(Iteration 2311 / 4900) loss: 1.126034
(Iteration 2321 / 4900) loss: 1.266319
(Iteration 2331 / 4900) loss: 1.063174
(Iteration 2341 / 4900) loss: 1.425850
(Iteration 2351 / 4900) loss: 1.112710
(Iteration 2361 / 4900) loss: 1.351412
(Iteration 2371 / 4900) loss: 1.324595
(Iteration 2381 / 4900) loss: 1.270101
(Iteration 2391 / 4900) loss: 1.235389
(Iteration 2401 / 4900) loss: 1.243494
(Iteration 2411 / 4900) loss: 1.481802
(Iteration 2421 / 4900) loss: 1.235206
(Iteration 2431 / 4900) loss: 1.168022
(Iteration 2441 / 4900) loss: 1.196861
(Epoch 5 / 10) train acc: 0.563000; val_acc: 0.485000
(Iteration 2451 / 4900) loss: 1.204262
(Iteration 2461 / 4900) loss: 1.040710
(Iteration 2471 / 4900) loss: 1.165884
(Iteration 2481 / 4900) loss: 1.210631
(Iteration 2491 / 4900) loss: 1.044842
(Iteration 2501 / 4900) loss: 1.391242
(Iteration 2511 / 4900) loss: 1.328672
(Iteration 2521 / 4900) loss: 1.220911
(Iteration 2531 / 4900) loss: 1.214891
(Iteration 2541 / 4900) loss: 1.244215
(Iteration 2551 / 4900) loss: 1.347471
(Iteration 2561 / 4900) loss: 1.256070
(Iteration 2571 / 4900) loss: 1.233943
(Iteration 2581 / 4900) loss: 1.314093
(Iteration 2591 / 4900) loss: 1.267555
(Iteration 2601 / 4900) loss: 1.392515
(Iteration 2611 / 4900) loss: 1.278381
```

```
(Iteration 2621 / 4900) loss: 1.182971
(Iteration 2631 / 4900) loss: 1.482361
(Iteration 2641 / 4900) loss: 1.217843
(Iteration 2651 / 4900) loss: 1.214005
(Iteration 2661 / 4900) loss: 1.305108
(Iteration 2671 / 4900) loss: 1.063529
(Iteration 2681 / 4900) loss: 1.118788
(Iteration 2691 / 4900) loss: 1.260244
(Iteration 2701 / 4900) loss: 1.522426
(Iteration 2711 / 4900) loss: 1.346427
(Iteration 2721 / 4900) loss: 1.202664
(Iteration 2731 / 4900) loss: 1.042910
(Iteration 2741 / 4900) loss: 1.267010
(Iteration 2751 / 4900) loss: 1.245915
(Iteration 2761 / 4900) loss: 1.234488
(Iteration 2771 / 4900) loss: 1.174539
(Iteration 2781 / 4900) loss: 1.045108
(Iteration 2791 / 4900) loss: 1.105482
(Iteration 2801 / 4900) loss: 1.117375
(Iteration 2811 / 4900) loss: 1.145276
(Iteration 2821 / 4900) loss: 1.463906
(Iteration 2831 / 4900) loss: 0.989979
(Iteration 2841 / 4900) loss: 1.166490
(Iteration 2851 / 4900) loss: 1.317097
(Iteration 2861 / 4900) loss: 1.279989
(Iteration 2871 / 4900) loss: 1.180661
(Iteration 2881 / 4900) loss: 1.178023
(Iteration 2891 / 4900) loss: 1.167373
(Iteration 2901 / 4900) loss: 0.978691
(Iteration 2911 / 4900) loss: 1.147971
(Iteration 2921 / 4900) loss: 1.236062
(Iteration 2931 / 4900) loss: 1.299416
(Epoch 6 / 10) train acc: 0.606000; val_acc: 0.507000
(Iteration 2941 / 4900) loss: 1.187989
(Iteration 2951 / 4900) loss: 1.337760
(Iteration 2961 / 4900) loss: 1.066255
(Iteration 2971 / 4900) loss: 1.276732
(Iteration 2981 / 4900) loss: 1.224480
(Iteration 2991 / 4900) loss: 1.088217
(Iteration 3001 / 4900) loss: 1.103570
(Iteration 3011 / 4900) loss: 1.302228
(Iteration 3021 / 4900) loss: 1.227006
(Iteration 3031 / 4900) loss: 1.149238
(Iteration 3041 / 4900) loss: 1.089411
(Iteration 3051 / 4900) loss: 1.288844
(Iteration 3061 / 4900) loss: 1.243189
(Iteration 3071 / 4900) loss: 1.160734
(Iteration 3081 / 4900) loss: 1.155887
```

```
(Iteration 3091 / 4900) loss: 1.097665
(Iteration 3101 / 4900) loss: 1.218568
(Iteration 3111 / 4900) loss: 1.079645
(Iteration 3121 / 4900) loss: 1.432215
(Iteration 3131 / 4900) loss: 1.261826
(Iteration 3141 / 4900) loss: 1.243991
(Iteration 3151 / 4900) loss: 1.043579
(Iteration 3161 / 4900) loss: 1.234223
(Iteration 3171 / 4900) loss: 1.330154
(Iteration 3181 / 4900) loss: 1.094074
(Iteration 3191 / 4900) loss: 1.177350
(Iteration 3201 / 4900) loss: 1.239721
(Iteration 3211 / 4900) loss: 1.353119
(Iteration 3221 / 4900) loss: 1.200025
(Iteration 3231 / 4900) loss: 1.222660
(Iteration 3241 / 4900) loss: 1.354623
(Iteration 3251 / 4900) loss: 1.178167
(Iteration 3261 / 4900) loss: 1.044392
(Iteration 3271 / 4900) loss: 1.149214
(Iteration 3281 / 4900) loss: 0.888857
(Iteration 3291 / 4900) loss: 1.083028
(Iteration 3301 / 4900) loss: 1.203360
(Iteration 3311 / 4900) loss: 1.131248
(Iteration 3321 / 4900) loss: 1.143116
(Iteration 3331 / 4900) loss: 0.962991
(Iteration 3341 / 4900) loss: 1.235668
(Iteration 3351 / 4900) loss: 1.134243
(Iteration 3361 / 4900) loss: 1.304440
(Iteration 3371 / 4900) loss: 1.143076
(Iteration 3381 / 4900) loss: 1.061689
(Iteration 3391 / 4900) loss: 1.415506
(Iteration 3401 / 4900) loss: 1.058706
(Iteration 3411 / 4900) loss: 1.038444
(Iteration 3421 / 4900) loss: 1.201481
(Epoch 7 / 10) train acc: 0.581000; val_acc: 0.501000
(Iteration 3431 / 4900) loss: 1.389487
(Iteration 3441 / 4900) loss: 1.242735
(Iteration 3451 / 4900) loss: 1.206109
(Iteration 3461 / 4900) loss: 1.203064
(Iteration 3471 / 4900) loss: 1.215513
(Iteration 3481 / 4900) loss: 0.816967
(Iteration 3491 / 4900) loss: 1.311253
(Iteration 3501 / 4900) loss: 1.254037
(Iteration 3511 / 4900) loss: 1.266620
(Iteration 3521 / 4900) loss: 1.258641
(Iteration 3531 / 4900) loss: 1.126579
(Iteration 3541 / 4900) loss: 1.043645
(Iteration 3551 / 4900) loss: 0.998314
```

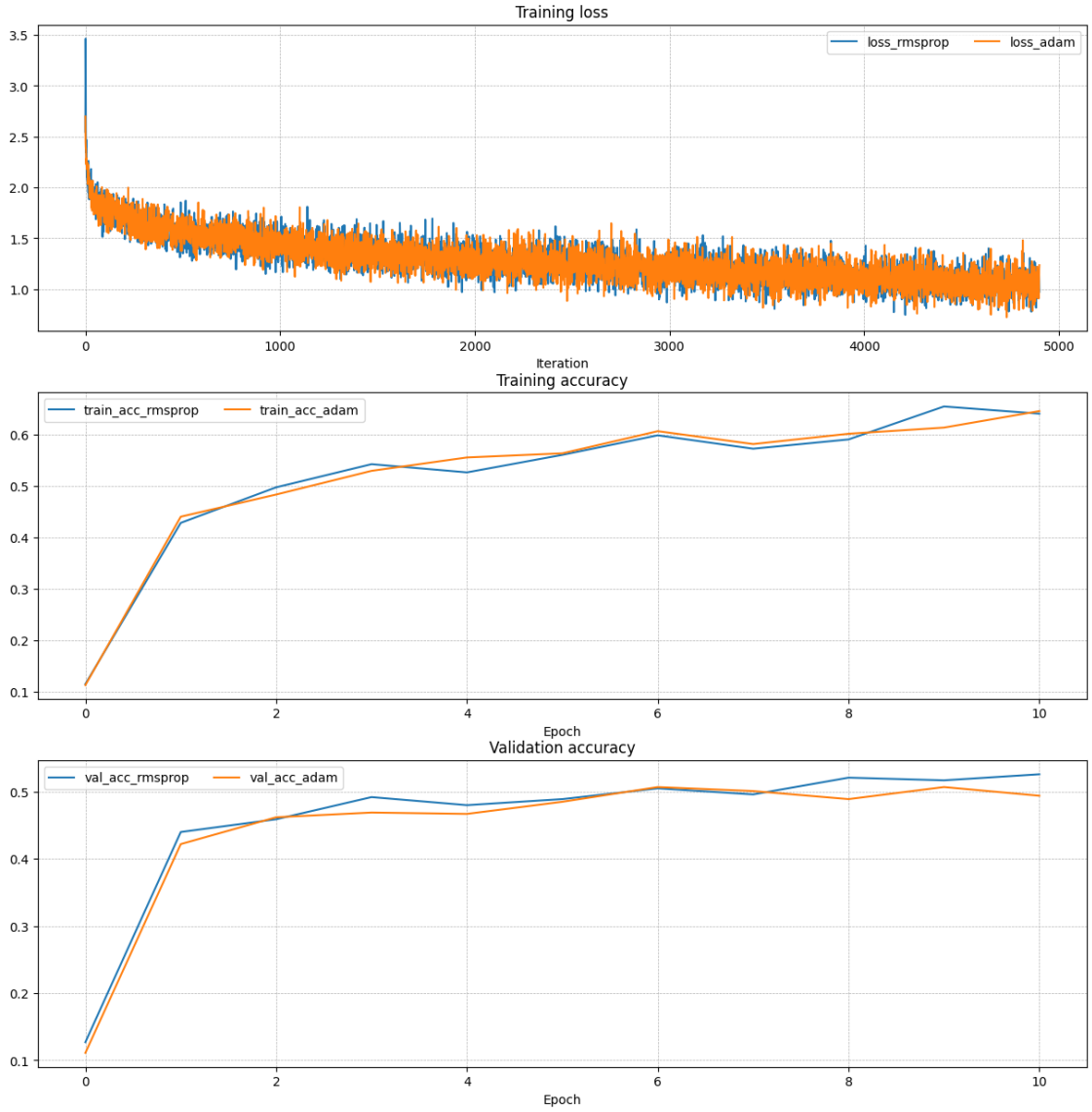
```
(Iteration 3561 / 4900) loss: 0.873637
(Iteration 3571 / 4900) loss: 1.105948
(Iteration 3581 / 4900) loss: 1.232063
(Iteration 3591 / 4900) loss: 1.096225
(Iteration 3601 / 4900) loss: 1.035737
(Iteration 3611 / 4900) loss: 1.102201
(Iteration 3621 / 4900) loss: 1.169913
(Iteration 3631 / 4900) loss: 1.085500
(Iteration 3641 / 4900) loss: 1.076725
(Iteration 3651 / 4900) loss: 0.893865
(Iteration 3661 / 4900) loss: 0.901283
(Iteration 3671 / 4900) loss: 1.070471
(Iteration 3681 / 4900) loss: 1.003623
(Iteration 3691 / 4900) loss: 1.071928
(Iteration 3701 / 4900) loss: 1.358161
(Iteration 3711 / 4900) loss: 1.048013
(Iteration 3721 / 4900) loss: 1.086530
(Iteration 3731 / 4900) loss: 1.062639
(Iteration 3741 / 4900) loss: 1.074635
(Iteration 3751 / 4900) loss: 1.249095
(Iteration 3761 / 4900) loss: 1.182548
(Iteration 3771 / 4900) loss: 1.153529
(Iteration 3781 / 4900) loss: 1.106366
(Iteration 3791 / 4900) loss: 1.105032
(Iteration 3801 / 4900) loss: 1.120161
(Iteration 3811 / 4900) loss: 1.060681
(Iteration 3821 / 4900) loss: 1.128040
(Iteration 3831 / 4900) loss: 1.145720
(Iteration 3841 / 4900) loss: 1.085775
(Iteration 3851 / 4900) loss: 1.134040
(Iteration 3861 / 4900) loss: 0.936363
(Iteration 3871 / 4900) loss: 1.221745
(Iteration 3881 / 4900) loss: 0.987962
(Iteration 3891 / 4900) loss: 1.086597
(Iteration 3901 / 4900) loss: 0.988391
(Iteration 3911 / 4900) loss: 1.062206
(Epoch 8 / 10) train acc: 0.601000; val_acc: 0.489000
(Iteration 3921 / 4900) loss: 0.973689
(Iteration 3931 / 4900) loss: 1.118688
(Iteration 3941 / 4900) loss: 1.180740
(Iteration 3951 / 4900) loss: 1.224028
(Iteration 3961 / 4900) loss: 1.212895
(Iteration 3971 / 4900) loss: 1.099633
(Iteration 3981 / 4900) loss: 1.287192
(Iteration 3991 / 4900) loss: 1.048927
(Iteration 4001 / 4900) loss: 1.055271
(Iteration 4011 / 4900) loss: 1.131947
(Iteration 4021 / 4900) loss: 0.979802
```

```
(Iteration 4031 / 4900) loss: 0.943846
(Iteration 4041 / 4900) loss: 0.866546
(Iteration 4051 / 4900) loss: 1.105139
(Iteration 4061 / 4900) loss: 0.858238
(Iteration 4071 / 4900) loss: 1.285307
(Iteration 4081 / 4900) loss: 1.136764
(Iteration 4091 / 4900) loss: 1.089048
(Iteration 4101 / 4900) loss: 1.207597
(Iteration 4111 / 4900) loss: 1.021105
(Iteration 4121 / 4900) loss: 1.043462
(Iteration 4131 / 4900) loss: 1.102009
(Iteration 4141 / 4900) loss: 1.171248
(Iteration 4151 / 4900) loss: 0.962684
(Iteration 4161 / 4900) loss: 1.195974
(Iteration 4171 / 4900) loss: 1.180849
(Iteration 4181 / 4900) loss: 1.122541
(Iteration 4191 / 4900) loss: 1.187090
(Iteration 4201 / 4900) loss: 0.912821
(Iteration 4211 / 4900) loss: 1.126359
(Iteration 4221 / 4900) loss: 1.089037
(Iteration 4231 / 4900) loss: 1.080542
(Iteration 4241 / 4900) loss: 0.982764
(Iteration 4251 / 4900) loss: 1.096832
(Iteration 4261 / 4900) loss: 1.017888
(Iteration 4271 / 4900) loss: 1.084970
(Iteration 4281 / 4900) loss: 1.011836
(Iteration 4291 / 4900) loss: 0.874615
(Iteration 4301 / 4900) loss: 1.026821
(Iteration 4311 / 4900) loss: 1.181454
(Iteration 4321 / 4900) loss: 1.089847
(Iteration 4331 / 4900) loss: 0.947589
(Iteration 4341 / 4900) loss: 1.008729
(Iteration 4351 / 4900) loss: 0.962146
(Iteration 4361 / 4900) loss: 0.823221
(Iteration 4371 / 4900) loss: 1.250174
(Iteration 4381 / 4900) loss: 0.996914
(Iteration 4391 / 4900) loss: 1.077787
(Iteration 4401 / 4900) loss: 1.141231
(Epoch 9 / 10) train acc: 0.613000; val_acc: 0.507000
(Iteration 4411 / 4900) loss: 1.148739
(Iteration 4421 / 4900) loss: 1.010237
(Iteration 4431 / 4900) loss: 1.166330
(Iteration 4441 / 4900) loss: 1.242155
(Iteration 4451 / 4900) loss: 0.868802
(Iteration 4461 / 4900) loss: 1.091891
(Iteration 4471 / 4900) loss: 1.037386
(Iteration 4481 / 4900) loss: 1.063515
(Iteration 4491 / 4900) loss: 0.948963
```

```
(Iteration 4501 / 4900) loss: 1.225216
(Iteration 4511 / 4900) loss: 1.074854
(Iteration 4521 / 4900) loss: 0.968069
(Iteration 4531 / 4900) loss: 0.970324
(Iteration 4541 / 4900) loss: 1.097583
(Iteration 4551 / 4900) loss: 1.086644
(Iteration 4561 / 4900) loss: 1.086005
(Iteration 4571 / 4900) loss: 1.222322
(Iteration 4581 / 4900) loss: 0.974423
(Iteration 4591 / 4900) loss: 1.045777
(Iteration 4601 / 4900) loss: 1.081614
(Iteration 4611 / 4900) loss: 1.071830
(Iteration 4621 / 4900) loss: 1.081941
(Iteration 4631 / 4900) loss: 0.921438
(Iteration 4641 / 4900) loss: 1.088766
(Iteration 4651 / 4900) loss: 0.748405
(Iteration 4661 / 4900) loss: 1.020171
(Iteration 4671 / 4900) loss: 0.847070
(Iteration 4681 / 4900) loss: 0.967879
(Iteration 4691 / 4900) loss: 0.995610
(Iteration 4701 / 4900) loss: 1.274464
(Iteration 4711 / 4900) loss: 0.890875
(Iteration 4721 / 4900) loss: 0.967537
(Iteration 4731 / 4900) loss: 1.255580
(Iteration 4741 / 4900) loss: 0.957572
(Iteration 4751 / 4900) loss: 1.024864
(Iteration 4761 / 4900) loss: 1.065401
(Iteration 4771 / 4900) loss: 0.806725
(Iteration 4781 / 4900) loss: 1.028308
(Iteration 4791 / 4900) loss: 0.943476
(Iteration 4801 / 4900) loss: 1.041714
(Iteration 4811 / 4900) loss: 1.131337
(Iteration 4821 / 4900) loss: 0.898525
(Iteration 4831 / 4900) loss: 1.178717
(Iteration 4841 / 4900) loss: 1.117728
(Iteration 4851 / 4900) loss: 1.055773
(Iteration 4861 / 4900) loss: 1.086916
(Iteration 4871 / 4900) loss: 0.942111
(Iteration 4881 / 4900) loss: 1.010538
(Iteration 4891 / 4900) loss: 0.912719
(Epoch 10 / 10) train acc: 0.645000; val_acc: 0.494000
best accuarcy: 0.544
```

[102]:
```
fig, axes = plt.subplots(3, 1, figsize=(15, 15))

axes[0].set_title('Training loss')
axes[0].set_xlabel('Iteration')
```

```
axes[1].set_title('Training accuracy')
axes[1].set_xlabel('Epoch')
axes[2].set_title('Validation accuracy')
axes[2].set_xlabel('Epoch')

for update_rule, solver in solvers.items():
    axes[0].plot(solver.loss_history, label=f"loss_{update_rule}")
    axes[1].plot(solver.train_acc_history, label=f"train_acc_{update_rule}")
    axes[2].plot(solver.val_acc_history, label=f"val_acc_{update_rule}")

for ax in axes:
    ax.legend(loc="best", ncol=4)
    ax.grid(linestyle='--', linewidth=0.5)

plt.show()
```



4 Test Your Model!

Run your best model on the validation and test sets. You should achieve at least 50% accuracy on the validation set.

[103]:
```
y_test_pred = np.argmax(best_model.loss(data['X_test']), axis=1)
y_val_pred = np.argmax(best_model.loss(data['X_val']), axis=1)
print('Validation set accuracy: ', (y_val_pred == data['y_val']).mean())
print('Test set accuracy: ', (y_test_pred == data['y_test']).mean())
```

```
Validation set accuracy:  0.544
Test set accuracy:  0.519
```