two_layer_net

January 31, 2024

[1]:
```
# 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.

[2]:
```
# 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))))
```

[3]:
```
# 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:

[4]:
```
# 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.

[5]:
```
# 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:

[6]:
```
# 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:

[7]:
```
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 gives gradients close to 0 at negative values less than -2 of the input. ReLu gives gradients exactly 0 at any negative value of the input.

Leaky RelU will never give a 0 gradient even if the inputs are negative.

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:

[8]:
```
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:

[9]:
```
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 sum_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:  8.998158338429791
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.

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

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


solver = Solver(model, data,
                update_rule='sgd',
                optim_config={
                  'learning_rate': 1e-4,
                },
                lr_decay=0.95,
                num_epochs=5, batch_size=200,
                print_every=100)
solver.train()

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

```
(Iteration 1 / 1225) loss: 2.300129
(Epoch 0 / 5) train acc: 0.138000; val_acc: 0.140000
(Iteration 101 / 1225) loss: 2.251986
(Iteration 201 / 1225) loss: 2.194247
(Epoch 1 / 5) train acc: 0.248000; val_acc: 0.255000
```

```
(Iteration 301 / 1225) loss: 2.100231
(Iteration 401 / 1225) loss: 2.047014
(Epoch 2 / 5) train acc: 0.310000; val_acc: 0.302000
(Iteration 501 / 1225) loss: 1.931189
(Iteration 601 / 1225) loss: 1.934842
(Iteration 701 / 1225) loss: 1.901075
(Epoch 3 / 5) train acc: 0.326000; val_acc: 0.326000
(Iteration 801 / 1225) loss: 1.818142
(Iteration 901 / 1225) loss: 1.979605
(Epoch 4 / 5) train acc: 0.345000; val_acc: 0.350000
(Iteration 1001 / 1225) loss: 1.887235
(Iteration 1101 / 1225) loss: 1.779911
(Iteration 1201 / 1225) loss: 1.723447
(Epoch 5 / 5) train acc: 0.347000; val_acc: 0.367000
```

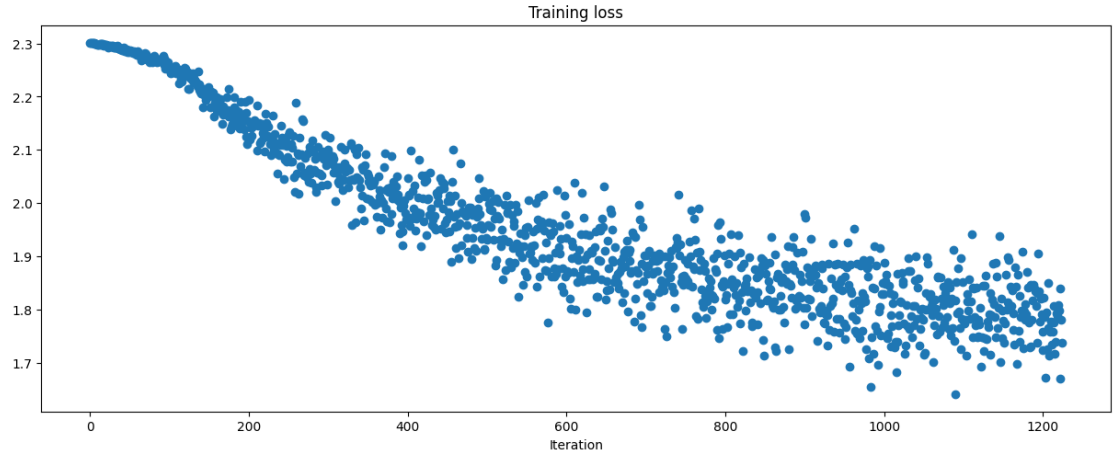
10 Debug the training

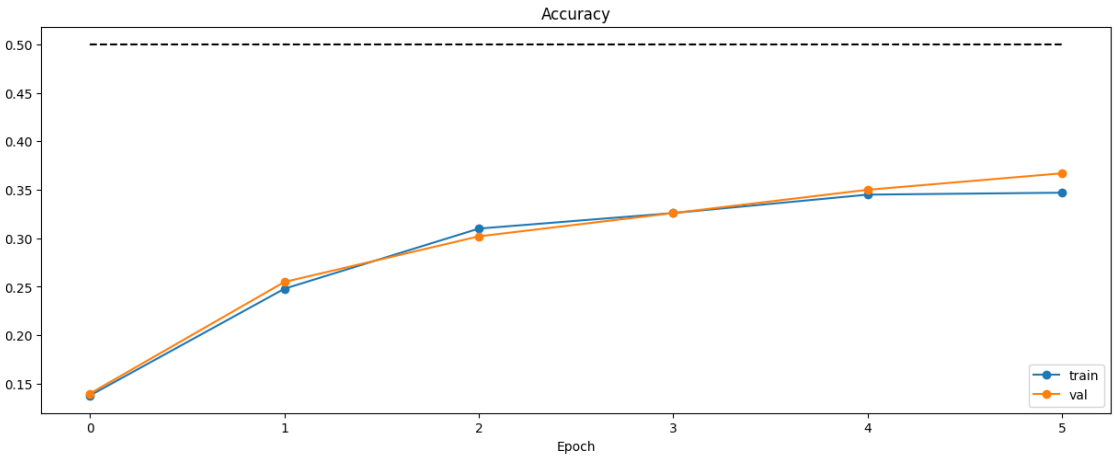
With the default parameters we provided above, you should get a validation accuracy of about 0.36 on the validation set. This isn't very good.

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

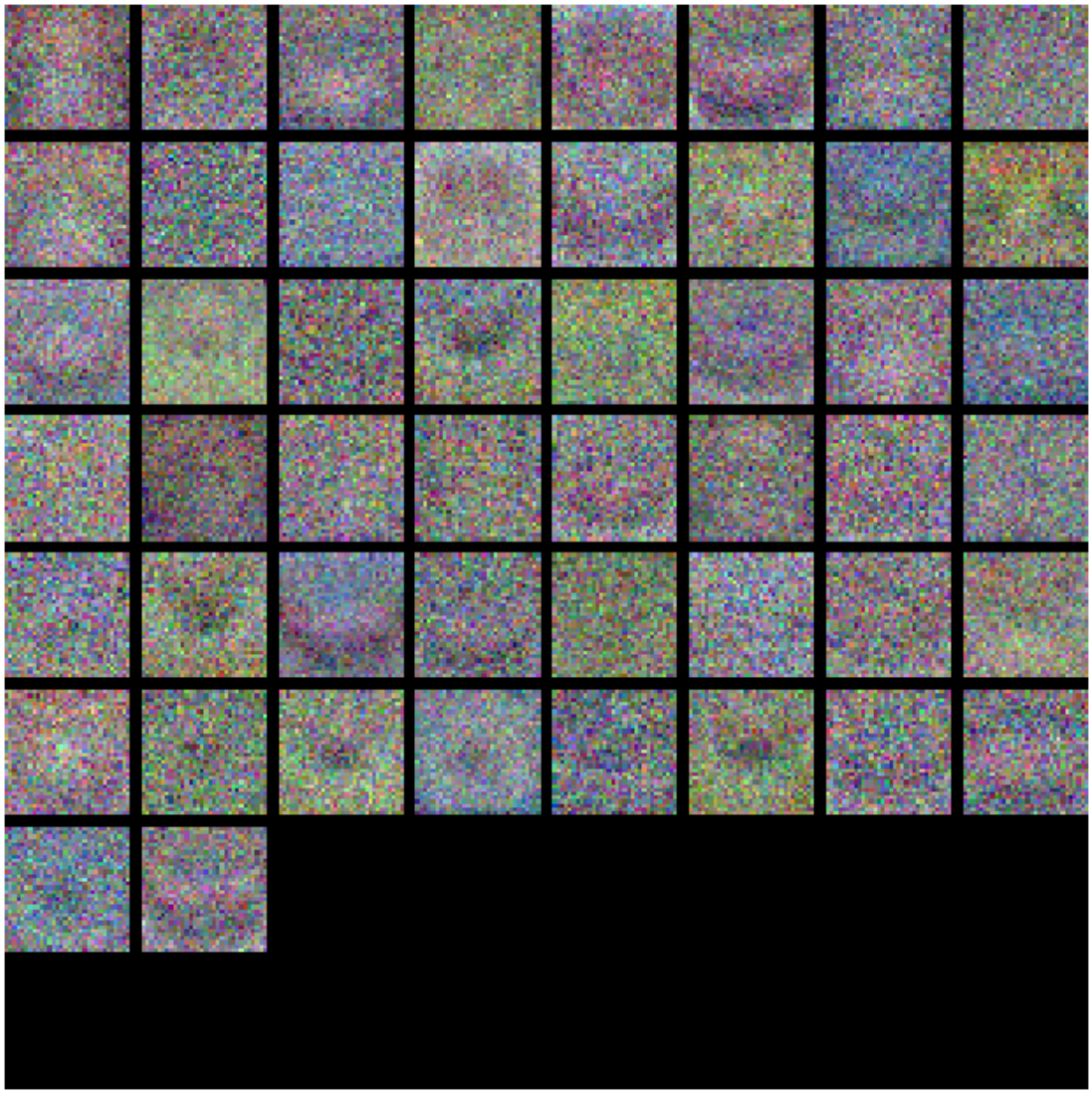
Another strategy is to visualize the weights that were learned in the first layer of the network. In most neural networks trained on visual data, the first layer weights typically show some visible structure when visualized.

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

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

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





[14]:
```python
from 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.).

[15]:
```
best_model = None


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

input_size = 32 * 32 * 3
hidden_size = 1000
num_classes = 10
model = TwoLayerNet(input_size, hidden_size, num_classes, reg = 3)

solver = Solver(model, data,
                update_rule='sgd',
                optim_config={
                  'learning_rate': 3e-4,
                },
```

```
                lr_decay=0.95,
                num_epochs=15, batch_size=200,
                print_every=100)
solver.train()

best_model = model

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

```
(Iteration 1 / 3675) loss: 6.926036
(Epoch 0 / 15) train acc: 0.148000; val_acc: 0.145000
(Iteration 101 / 3675) loss: 5.760590
(Iteration 201 / 3675) loss: 5.090103
(Epoch 1 / 15) train acc: 0.393000; val_acc: 0.397000
(Iteration 301 / 3675) loss: 4.415924
(Iteration 401 / 3675) loss: 3.996536
(Epoch 2 / 15) train acc: 0.419000; val_acc: 0.413000
(Iteration 501 / 3675) loss: 3.647049
(Iteration 601 / 3675) loss: 3.409784
(Iteration 701 / 3675) loss: 3.072206
(Epoch 3 / 15) train acc: 0.431000; val_acc: 0.443000
(Iteration 801 / 3675) loss: 2.920015
(Iteration 901 / 3675) loss: 2.722576
(Epoch 4 / 15) train acc: 0.465000; val_acc: 0.456000
(Iteration 1001 / 3675) loss: 2.590770
(Iteration 1101 / 3675) loss: 2.449275
(Iteration 1201 / 3675) loss: 2.356143
(Epoch 5 / 15) train acc: 0.456000; val_acc: 0.459000
(Iteration 1301 / 3675) loss: 2.311062
(Iteration 1401 / 3675) loss: 2.193227
(Epoch 6 / 15) train acc: 0.484000; val_acc: 0.463000
(Iteration 1501 / 3675) loss: 2.109608
(Iteration 1601 / 3675) loss: 2.109716
(Iteration 1701 / 3675) loss: 1.952872
(Epoch 7 / 15) train acc: 0.488000; val_acc: 0.471000
(Iteration 1801 / 3675) loss: 1.992479
(Iteration 1901 / 3675) loss: 2.026699
(Epoch 8 / 15) train acc: 0.481000; val_acc: 0.456000
(Iteration 2001 / 3675) loss: 2.007349
(Iteration 2101 / 3675) loss: 1.875870
(Iteration 2201 / 3675) loss: 1.789917
(Epoch 9 / 15) train acc: 0.481000; val_acc: 0.470000
(Iteration 2301 / 3675) loss: 1.838754
(Iteration 2401 / 3675) loss: 1.828313
```

```
(Epoch 10 / 15) train acc: 0.488000; val_acc: 0.474000
(Iteration 2501 / 3675) loss: 1.734419
(Iteration 2601 / 3675) loss: 1.790833
(Epoch 11 / 15) train acc: 0.495000; val_acc: 0.486000
(Iteration 2701 / 3675) loss: 1.729816
(Iteration 2801 / 3675) loss: 1.729894
(Iteration 2901 / 3675) loss: 1.801287
(Epoch 12 / 15) train acc: 0.507000; val_acc: 0.481000
(Iteration 3001 / 3675) loss: 1.842256
(Iteration 3101 / 3675) loss: 1.839068
(Epoch 13 / 15) train acc: 0.482000; val_acc: 0.479000
(Iteration 3201 / 3675) loss: 1.740956
(Iteration 3301 / 3675) loss: 1.816546
(Iteration 3401 / 3675) loss: 1.706868
(Epoch 14 / 15) train acc: 0.493000; val_acc: 0.478000
(Iteration 3501 / 3675) loss: 1.738893
(Iteration 3601 / 3675) loss: 1.638109
(Epoch 15 / 15) train acc: 0.514000; val_acc: 0.485000
```

12 Test your model!

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

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

```
Validation set accuracy:  0.486
```

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

```
Test set accuracy:  0.47
```

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 :

Training on a large dataset would include more potential variations in our training data to learn. Increasing regularization strength would prevent overfitting on the training dataset and hence make our model better suited to new information.

Adding more hidden units might not help with this problem as it would just increase overfitting on training data

features

January 31, 2024

[32]:
```
# 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 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.

[33]:
```
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
```

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

1.1 Load data

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

[34]:
```
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.

[37]:
```
from cse493g1.features import *

num_color_bins = 14 # Number of bins in the color histogram
feature_fns = [hog_feature, lambda img: color_histogram_hsv(img,␣
 ↪nbin=num_color_bins)]
X_train_feats = extract_features(X_train, feature_fns, verbose=True)
X_val_feats = extract_features(X_val, feature_fns)
X_test_feats = extract_features(X_test, feature_fns)

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

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

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

```
X_val_feats = np.hstack([X_val_feats, np.ones((X_val_feats.shape[0], 1))])
X_test_feats = np.hstack([X_test_feats, np.ones((X_test_feats.shape[0], 1))])
```

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

```
Done extracting features for 45000 / 49000 images
Done extracting features for 46000 / 49000 images
Done extracting features for 47000 / 49000 images
Done extracting features for 48000 / 49000 images
Done extracting features for 49000 / 49000 images
```

1.3 Train SVM on features

Using the multiclass SVM code developed 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.

[39]:
```
# 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, 2.5e-7, 1e-7]
regularization_strengths = [5e4, 2.5e5, 5e5, 2.5e6, 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)*****

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

    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)

    if val_accuracy > best_val:
      best_val = val_accuracy
```

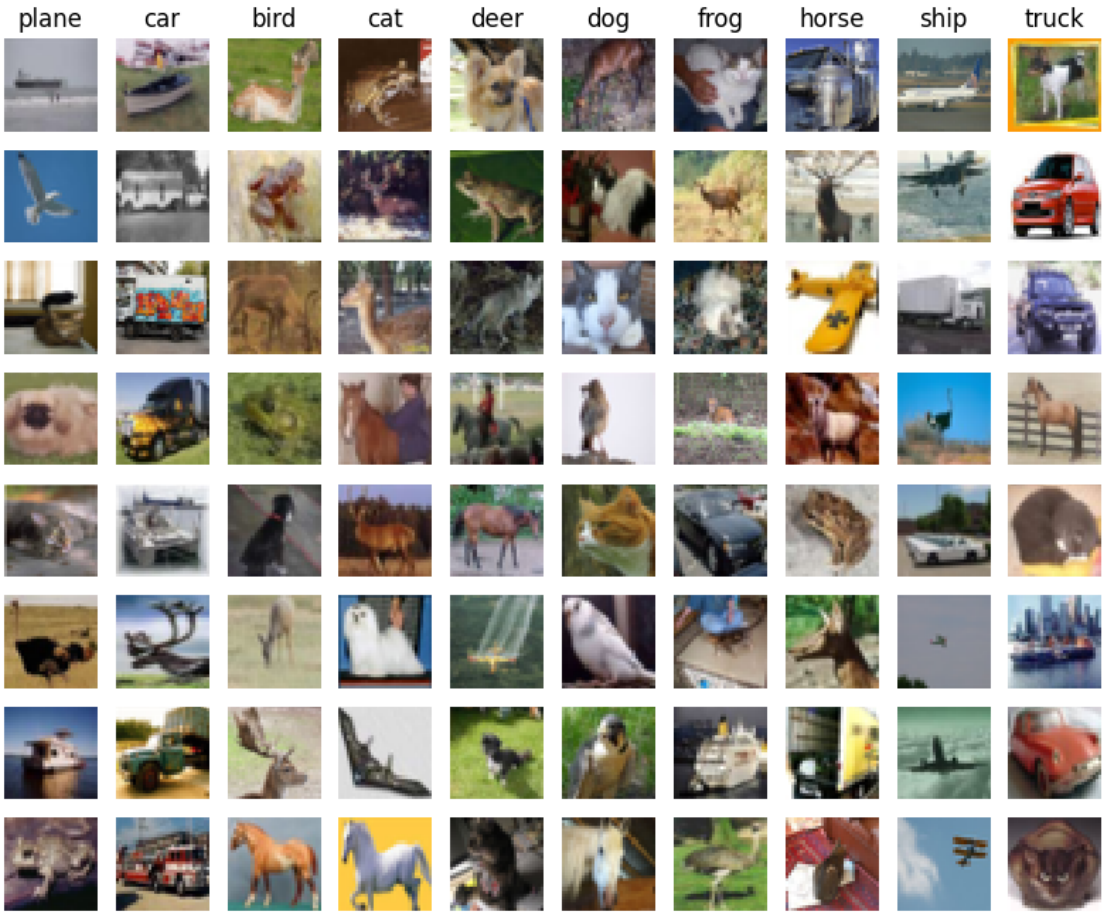
```
      best_svm = svm

    results[(learn_rate,reg_strength)] = (train_accuracy, val_accuracy)


# *****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)
```

```
/content/drive/My
Drive/cse493g1/assignments/assignment2/cse493g1/classifiers/linear_svm.py:97:
RuntimeWarning: overflow encountered in double_scalars
  loss = loss/num_train + reg * np.sum(W * W)
/usr/local/lib/python3.10/dist-packages/numpy/core/fromnumeric.py:86:
RuntimeWarning: overflow encountered in reduce
  return ufunc.reduce(obj, axis, dtype, out, **passkwargs)
/content/drive/My
Drive/cse493g1/assignments/assignment2/cse493g1/classifiers/linear_svm.py:97:
RuntimeWarning: overflow encountered in multiply
  loss = loss/num_train + reg * np.sum(W * W)
lr 1.000000e-09 reg 5.000000e+04 train accuracy: 0.092469 val accuracy: 0.095000
lr 1.000000e-09 reg 2.500000e+05 train accuracy: 0.095918 val accuracy: 0.097000
lr 1.000000e-09 reg 5.000000e+05 train accuracy: 0.095163 val accuracy: 0.079000
lr 1.000000e-09 reg 2.500000e+06 train accuracy: 0.155000 val accuracy: 0.156000
lr 1.000000e-09 reg 5.000000e+06 train accuracy: 0.417449 val accuracy: 0.418000
lr 1.000000e-08 reg 5.000000e+04 train accuracy: 0.093531 val accuracy: 0.101000
lr 1.000000e-08 reg 2.500000e+05 train accuracy: 0.352959 val accuracy: 0.339000
lr 1.000000e-08 reg 5.000000e+05 train accuracy: 0.417633 val accuracy: 0.421000
lr 1.000000e-08 reg 2.500000e+06 train accuracy: 0.413286 val accuracy: 0.428000
lr 1.000000e-08 reg 5.000000e+06 train accuracy: 0.403653 val accuracy: 0.387000
lr 1.000000e-07 reg 5.000000e+04 train accuracy: 0.418776 val accuracy: 0.434000
lr 1.000000e-07 reg 2.500000e+05 train accuracy: 0.419000 val accuracy: 0.414000
lr 1.000000e-07 reg 5.000000e+05 train accuracy: 0.419816 val accuracy: 0.421000
lr 1.000000e-07 reg 2.500000e+06 train accuracy: 0.374816 val accuracy: 0.375000
lr 1.000000e-07 reg 5.000000e+06 train accuracy: 0.342714 val accuracy: 0.329000
lr 2.500000e-07 reg 5.000000e+04 train accuracy: 0.407980 val accuracy: 0.401000
lr 2.500000e-07 reg 2.500000e+05 train accuracy: 0.412224 val accuracy: 0.415000
lr 2.500000e-07 reg 5.000000e+05 train accuracy: 0.390367 val accuracy: 0.400000
lr 2.500000e-07 reg 2.500000e+06 train accuracy: 0.303163 val accuracy: 0.304000
lr 2.500000e-07 reg 5.000000e+06 train accuracy: 0.088878 val accuracy: 0.098000
best validation accuracy achieved: 0.434000
```

[40]:
```
# 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)
```

```
0.422
```

[41]:
```
# 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 results seem to be very random with no identifiable pixel patterns. I cannot make any sense of it

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.

[42]:
```
# 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, 159)
(49000, 158)
```

[44]:
```
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 = [1e-2, 3e-2, 1e-1]
regularization_strengths = [0.5e-2, 1e-2, 1e-3]
hidden_size = [500]

num_classes = 10

results = {}
best_val = -1
```

```
for hidden_dim in hidden_size:
  for learn_rate in learning_rates:
    for reg_strength in regularization_strengths:
      net = TwoLayerNet(input_dim, hidden_dim, num_classes, reg = reg_strength)
      solver = Solver(net, data,
                update_rule='sgd',
                optim_config={
                  'learning_rate': learn_rate,
                },
                lr_decay=0.95,
                num_epochs=15, batch_size=200,
                print_every=100)
      solver.train()

      if solver.best_val_acc > best_val:
        best_net = net
        best_val = solver.best_val_acc

      results[(learn_rate,reg_strength,hidden_dim)] = (solver.
  ↪train_acc_history[len(solver.train_acc_history) - 1], solver.best_val_acc)

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

print('best validation accuracy achieved: %f' % best_val)

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

```
(Iteration 1 / 3675) loss: 2.302798
(Epoch 0 / 15) train acc: 0.107000; val_acc: 0.116000
(Iteration 101 / 3675) loss: 2.302560
(Iteration 201 / 3675) loss: 2.301855
(Epoch 1 / 15) train acc: 0.151000; val_acc: 0.116000
(Iteration 301 / 3675) loss: 2.301760
(Iteration 401 / 3675) loss: 2.301044
(Epoch 2 / 15) train acc: 0.235000; val_acc: 0.231000
(Iteration 501 / 3675) loss: 2.300080
(Iteration 601 / 3675) loss: 2.298758
(Iteration 701 / 3675) loss: 2.294825
(Epoch 3 / 15) train acc: 0.301000; val_acc: 0.273000
(Iteration 801 / 3675) loss: 2.290097
(Iteration 901 / 3675) loss: 2.280606
(Epoch 4 / 15) train acc: 0.318000; val_acc: 0.270000
```

```
(Iteration 1001 / 3675) loss: 2.272445
(Iteration 1101 / 3675) loss: 2.241707
(Iteration 1201 / 3675) loss: 2.224672
(Epoch 5 / 15) train acc: 0.277000; val_acc: 0.290000
(Iteration 1301 / 3675) loss: 2.129173
(Iteration 1401 / 3675) loss: 2.151524
(Epoch 6 / 15) train acc: 0.293000; val_acc: 0.280000
(Iteration 1501 / 3675) loss: 2.067216
(Iteration 1601 / 3675) loss: 2.002253
(Iteration 1701 / 3675) loss: 2.002806
(Epoch 7 / 15) train acc: 0.309000; val_acc: 0.312000
(Iteration 1801 / 3675) loss: 1.973844
(Iteration 1901 / 3675) loss: 1.962856
(Epoch 8 / 15) train acc: 0.314000; val_acc: 0.330000
(Iteration 2001 / 3675) loss: 1.909557
(Iteration 2101 / 3675) loss: 1.934415
(Iteration 2201 / 3675) loss: 1.818152
(Epoch 9 / 15) train acc: 0.339000; val_acc: 0.341000
(Iteration 2301 / 3675) loss: 1.866324
(Iteration 2401 / 3675) loss: 1.733348
(Epoch 10 / 15) train acc: 0.350000; val_acc: 0.360000
(Iteration 2501 / 3675) loss: 1.798570
(Iteration 2601 / 3675) loss: 1.814270
(Epoch 11 / 15) train acc: 0.390000; val_acc: 0.389000
(Iteration 2701 / 3675) loss: 1.722921
(Iteration 2801 / 3675) loss: 1.780900
(Iteration 2901 / 3675) loss: 1.694323
(Epoch 12 / 15) train acc: 0.411000; val_acc: 0.395000
(Iteration 3001 / 3675) loss: 1.687224
(Iteration 3101 / 3675) loss: 1.666330
(Epoch 13 / 15) train acc: 0.402000; val_acc: 0.397000
(Iteration 3201 / 3675) loss: 1.691008
(Iteration 3301 / 3675) loss: 1.751348
(Iteration 3401 / 3675) loss: 1.586062
(Epoch 14 / 15) train acc: 0.394000; val_acc: 0.404000
(Iteration 3501 / 3675) loss: 1.724275
(Iteration 3601 / 3675) loss: 1.566823
(Epoch 15 / 15) train acc: 0.398000; val_acc: 0.414000
(Iteration 1 / 3675) loss: 2.303025
(Epoch 0 / 15) train acc: 0.076000; val_acc: 0.079000
(Iteration 101 / 3675) loss: 2.302808
(Iteration 201 / 3675) loss: 2.302392
(Epoch 1 / 15) train acc: 0.097000; val_acc: 0.089000
(Iteration 301 / 3675) loss: 2.302202
(Iteration 401 / 3675) loss: 2.301345
(Epoch 2 / 15) train acc: 0.098000; val_acc: 0.088000
(Iteration 501 / 3675) loss: 2.300301
(Iteration 601 / 3675) loss: 2.298455
```

```
(Iteration 701 / 3675) loss: 2.295540
(Epoch 3 / 15) train acc: 0.182000; val_acc: 0.161000
(Iteration 801 / 3675) loss: 2.285902
(Iteration 901 / 3675) loss: 2.284033
(Epoch 4 / 15) train acc: 0.265000; val_acc: 0.227000
(Iteration 1001 / 3675) loss: 2.267490
(Iteration 1101 / 3675) loss: 2.239347
(Iteration 1201 / 3675) loss: 2.200114
(Epoch 5 / 15) train acc: 0.270000; val_acc: 0.270000
(Iteration 1301 / 3675) loss: 2.175351
(Iteration 1401 / 3675) loss: 2.112448
(Epoch 6 / 15) train acc: 0.274000; val_acc: 0.271000
(Iteration 1501 / 3675) loss: 2.101717
(Iteration 1601 / 3675) loss: 2.065878
(Iteration 1701 / 3675) loss: 2.024356
(Epoch 7 / 15) train acc: 0.267000; val_acc: 0.291000
(Iteration 1801 / 3675) loss: 2.024256
(Iteration 1901 / 3675) loss: 1.911374
(Epoch 8 / 15) train acc: 0.300000; val_acc: 0.316000
(Iteration 2001 / 3675) loss: 1.927283
(Iteration 2101 / 3675) loss: 1.901794
(Iteration 2201 / 3675) loss: 1.928822
(Epoch 9 / 15) train acc: 0.343000; val_acc: 0.336000
(Iteration 2301 / 3675) loss: 1.838562
(Iteration 2401 / 3675) loss: 1.783768
(Epoch 10 / 15) train acc: 0.353000; val_acc: 0.359000
(Iteration 2501 / 3675) loss: 1.781327
(Iteration 2601 / 3675) loss: 1.842835
(Epoch 11 / 15) train acc: 0.358000; val_acc: 0.376000
(Iteration 2701 / 3675) loss: 1.832639
(Iteration 2801 / 3675) loss: 1.854299
(Iteration 2901 / 3675) loss: 1.796189
(Epoch 12 / 15) train acc: 0.397000; val_acc: 0.386000
(Iteration 3001 / 3675) loss: 1.744981
(Iteration 3101 / 3675) loss: 1.728943
(Epoch 13 / 15) train acc: 0.393000; val_acc: 0.399000
(Iteration 3201 / 3675) loss: 1.763058
(Iteration 3301 / 3675) loss: 1.697440
(Iteration 3401 / 3675) loss: 1.645852
(Epoch 14 / 15) train acc: 0.402000; val_acc: 0.396000
(Iteration 3501 / 3675) loss: 1.703989
(Iteration 3601 / 3675) loss: 1.684553
(Epoch 15 / 15) train acc: 0.432000; val_acc: 0.409000
(Iteration 1 / 3675) loss: 2.302641
(Epoch 0 / 15) train acc: 0.103000; val_acc: 0.090000
(Iteration 101 / 3675) loss: 2.302405
(Iteration 201 / 3675) loss: 2.301943
(Epoch 1 / 15) train acc: 0.111000; val_acc: 0.133000
```

```
(Iteration 301 / 3675) loss: 2.302380
(Iteration 401 / 3675) loss: 2.301190
(Epoch 2 / 15) train acc: 0.292000; val_acc: 0.295000
(Iteration 501 / 3675) loss: 2.300072
(Iteration 601 / 3675) loss: 2.297906
(Iteration 701 / 3675) loss: 2.294390
(Epoch 3 / 15) train acc: 0.258000; val_acc: 0.267000
(Iteration 801 / 3675) loss: 2.287844
(Iteration 901 / 3675) loss: 2.275172
(Epoch 4 / 15) train acc: 0.261000; val_acc: 0.266000
(Iteration 1001 / 3675) loss: 2.256879
(Iteration 1101 / 3675) loss: 2.239894
(Iteration 1201 / 3675) loss: 2.206484
(Epoch 5 / 15) train acc: 0.264000; val_acc: 0.255000
(Iteration 1301 / 3675) loss: 2.154496
(Iteration 1401 / 3675) loss: 2.100276
(Epoch 6 / 15) train acc: 0.260000; val_acc: 0.270000
(Iteration 1501 / 3675) loss: 2.106406
(Iteration 1601 / 3675) loss: 2.082070
(Iteration 1701 / 3675) loss: 1.955235
(Epoch 7 / 15) train acc: 0.284000; val_acc: 0.301000
(Iteration 1801 / 3675) loss: 1.952929
(Iteration 1901 / 3675) loss: 1.885850
(Epoch 8 / 15) train acc: 0.323000; val_acc: 0.330000
(Iteration 2001 / 3675) loss: 1.855002
(Iteration 2101 / 3675) loss: 1.841808
(Iteration 2201 / 3675) loss: 1.851696
(Epoch 9 / 15) train acc: 0.343000; val_acc: 0.343000
(Iteration 2301 / 3675) loss: 1.828103
(Iteration 2401 / 3675) loss: 1.802353
(Epoch 10 / 15) train acc: 0.363000; val_acc: 0.354000
(Iteration 2501 / 3675) loss: 1.800113
(Iteration 2601 / 3675) loss: 1.716435
(Epoch 11 / 15) train acc: 0.381000; val_acc: 0.375000
(Iteration 2701 / 3675) loss: 1.692410
(Iteration 2801 / 3675) loss: 1.697879
(Iteration 2901 / 3675) loss: 1.688026
(Epoch 12 / 15) train acc: 0.399000; val_acc: 0.393000
(Iteration 3001 / 3675) loss: 1.733506
(Iteration 3101 / 3675) loss: 1.691115
(Epoch 13 / 15) train acc: 0.403000; val_acc: 0.408000
(Iteration 3201 / 3675) loss: 1.660354
(Iteration 3301 / 3675) loss: 1.705229
(Iteration 3401 / 3675) loss: 1.566874
(Epoch 14 / 15) train acc: 0.403000; val_acc: 0.406000
(Iteration 3501 / 3675) loss: 1.664710
(Iteration 3601 / 3675) loss: 1.435491
(Epoch 15 / 15) train acc: 0.426000; val_acc: 0.415000
```

```
(Iteration 1 / 3675) loss: 2.302784
(Epoch 0 / 15) train acc: 0.103000; val_acc: 0.081000
(Iteration 101 / 3675) loss: 2.302276
(Iteration 201 / 3675) loss: 2.296965
(Epoch 1 / 15) train acc: 0.220000; val_acc: 0.249000
(Iteration 301 / 3675) loss: 2.262573
(Iteration 401 / 3675) loss: 2.173058
(Epoch 2 / 15) train acc: 0.309000; val_acc: 0.303000
(Iteration 501 / 3675) loss: 1.989042
(Iteration 601 / 3675) loss: 1.893977
(Iteration 701 / 3675) loss: 1.792277
(Epoch 3 / 15) train acc: 0.383000; val_acc: 0.377000
(Iteration 801 / 3675) loss: 1.709936
(Iteration 901 / 3675) loss: 1.722994
(Epoch 4 / 15) train acc: 0.427000; val_acc: 0.411000
(Iteration 1001 / 3675) loss: 1.641564
(Iteration 1101 / 3675) loss: 1.587060
(Iteration 1201 / 3675) loss: 1.495033
(Epoch 5 / 15) train acc: 0.440000; val_acc: 0.450000
(Iteration 1301 / 3675) loss: 1.550185
(Iteration 1401 / 3675) loss: 1.592536
(Epoch 6 / 15) train acc: 0.506000; val_acc: 0.479000
(Iteration 1501 / 3675) loss: 1.479097
(Iteration 1601 / 3675) loss: 1.406112
(Iteration 1701 / 3675) loss: 1.410767
(Epoch 7 / 15) train acc: 0.481000; val_acc: 0.503000
(Iteration 1801 / 3675) loss: 1.413154
(Iteration 1901 / 3675) loss: 1.530556
(Epoch 8 / 15) train acc: 0.490000; val_acc: 0.513000
(Iteration 2001 / 3675) loss: 1.460687
(Iteration 2101 / 3675) loss: 1.477368
(Iteration 2201 / 3675) loss: 1.577411
(Epoch 9 / 15) train acc: 0.519000; val_acc: 0.521000
(Iteration 2301 / 3675) loss: 1.497063
(Iteration 2401 / 3675) loss: 1.464307
(Epoch 10 / 15) train acc: 0.532000; val_acc: 0.525000
(Iteration 2501 / 3675) loss: 1.490950
(Iteration 2601 / 3675) loss: 1.419676
(Epoch 11 / 15) train acc: 0.534000; val_acc: 0.522000
(Iteration 2701 / 3675) loss: 1.468359
(Iteration 2801 / 3675) loss: 1.376827
(Iteration 2901 / 3675) loss: 1.398966
(Epoch 12 / 15) train acc: 0.565000; val_acc: 0.519000
(Iteration 3001 / 3675) loss: 1.404273
(Iteration 3101 / 3675) loss: 1.428863
(Epoch 13 / 15) train acc: 0.518000; val_acc: 0.520000
(Iteration 3201 / 3675) loss: 1.397722
(Iteration 3301 / 3675) loss: 1.466221
```

```
(Iteration 3401 / 3675) loss: 1.352209
(Epoch 14 / 15) train acc: 0.532000; val_acc: 0.517000
(Iteration 3501 / 3675) loss: 1.394309
(Iteration 3601 / 3675) loss: 1.306798
(Epoch 15 / 15) train acc: 0.524000; val_acc: 0.521000
(Iteration 1 / 3675) loss: 2.303006
(Epoch 0 / 15) train acc: 0.104000; val_acc: 0.121000
(Iteration 101 / 3675) loss: 2.301428
(Iteration 201 / 3675) loss: 2.296814
(Epoch 1 / 15) train acc: 0.234000; val_acc: 0.205000
(Iteration 301 / 3675) loss: 2.260409
(Iteration 401 / 3675) loss: 2.150822
(Epoch 2 / 15) train acc: 0.295000; val_acc: 0.294000
(Iteration 501 / 3675) loss: 2.049108
(Iteration 601 / 3675) loss: 1.927300
(Iteration 701 / 3675) loss: 1.827108
(Epoch 3 / 15) train acc: 0.350000; val_acc: 0.370000
(Iteration 801 / 3675) loss: 1.718361
(Iteration 901 / 3675) loss: 1.657672
(Epoch 4 / 15) train acc: 0.420000; val_acc: 0.411000
(Iteration 1001 / 3675) loss: 1.629090
(Iteration 1101 / 3675) loss: 1.573151
(Iteration 1201 / 3675) loss: 1.617974
(Epoch 5 / 15) train acc: 0.429000; val_acc: 0.456000
(Iteration 1301 / 3675) loss: 1.533678
(Iteration 1401 / 3675) loss: 1.404574
(Epoch 6 / 15) train acc: 0.484000; val_acc: 0.477000
(Iteration 1501 / 3675) loss: 1.525320
(Iteration 1601 / 3675) loss: 1.483015
(Iteration 1701 / 3675) loss: 1.349152
(Epoch 7 / 15) train acc: 0.499000; val_acc: 0.488000
(Iteration 1801 / 3675) loss: 1.427785
(Iteration 1901 / 3675) loss: 1.440677
(Epoch 8 / 15) train acc: 0.509000; val_acc: 0.499000
(Iteration 2001 / 3675) loss: 1.445535
(Iteration 2101 / 3675) loss: 1.590298
(Iteration 2201 / 3675) loss: 1.510409
(Epoch 9 / 15) train acc: 0.546000; val_acc: 0.508000
(Iteration 2301 / 3675) loss: 1.500791
(Iteration 2401 / 3675) loss: 1.432687
(Epoch 10 / 15) train acc: 0.507000; val_acc: 0.513000
(Iteration 2501 / 3675) loss: 1.454675
(Iteration 2601 / 3675) loss: 1.504958
(Epoch 11 / 15) train acc: 0.507000; val_acc: 0.514000
(Iteration 2701 / 3675) loss: 1.468393
(Iteration 2801 / 3675) loss: 1.363659
(Iteration 2901 / 3675) loss: 1.606764
(Epoch 12 / 15) train acc: 0.530000; val_acc: 0.520000
```

```
(Iteration 3001 / 3675) loss: 1.480530
(Iteration 3101 / 3675) loss: 1.570226
(Epoch 13 / 15) train acc: 0.523000; val_acc: 0.517000
(Iteration 3201 / 3675) loss: 1.399918
(Iteration 3301 / 3675) loss: 1.626678
(Iteration 3401 / 3675) loss: 1.477758
(Epoch 14 / 15) train acc: 0.508000; val_acc: 0.520000
(Iteration 3501 / 3675) loss: 1.462489
(Iteration 3601 / 3675) loss: 1.274463
(Epoch 15 / 15) train acc: 0.534000; val_acc: 0.521000
(Iteration 1 / 3675) loss: 2.302638
(Epoch 0 / 15) train acc: 0.096000; val_acc: 0.087000
(Iteration 101 / 3675) loss: 2.302318
(Iteration 201 / 3675) loss: 2.298163
(Epoch 1 / 15) train acc: 0.279000; val_acc: 0.274000
(Iteration 301 / 3675) loss: 2.269060
(Iteration 401 / 3675) loss: 2.142579
(Epoch 2 / 15) train acc: 0.294000; val_acc: 0.305000
(Iteration 501 / 3675) loss: 2.001874
(Iteration 601 / 3675) loss: 1.854925
(Iteration 701 / 3675) loss: 1.753073
(Epoch 3 / 15) train acc: 0.381000; val_acc: 0.382000
(Iteration 801 / 3675) loss: 1.713411
(Iteration 901 / 3675) loss: 1.618314
(Epoch 4 / 15) train acc: 0.410000; val_acc: 0.424000
(Iteration 1001 / 3675) loss: 1.632746
(Iteration 1101 / 3675) loss: 1.531391
(Iteration 1201 / 3675) loss: 1.514970
(Epoch 5 / 15) train acc: 0.470000; val_acc: 0.455000
(Iteration 1301 / 3675) loss: 1.354388
(Iteration 1401 / 3675) loss: 1.624162
(Epoch 6 / 15) train acc: 0.489000; val_acc: 0.484000
(Iteration 1501 / 3675) loss: 1.534908
(Iteration 1601 / 3675) loss: 1.516861
(Iteration 1701 / 3675) loss: 1.470368
(Epoch 7 / 15) train acc: 0.474000; val_acc: 0.492000
(Iteration 1801 / 3675) loss: 1.453218
(Iteration 1901 / 3675) loss: 1.356684
(Epoch 8 / 15) train acc: 0.509000; val_acc: 0.507000
(Iteration 2001 / 3675) loss: 1.459717
(Iteration 2101 / 3675) loss: 1.350927
(Iteration 2201 / 3675) loss: 1.499891
(Epoch 9 / 15) train acc: 0.530000; val_acc: 0.510000
(Iteration 2301 / 3675) loss: 1.510328
(Iteration 2401 / 3675) loss: 1.365607
(Epoch 10 / 15) train acc: 0.510000; val_acc: 0.509000
(Iteration 2501 / 3675) loss: 1.367078
(Iteration 2601 / 3675) loss: 1.406137
```

```
(Epoch 11 / 15) train acc: 0.552000; val_acc: 0.519000
(Iteration 2701 / 3675) loss: 1.326691
(Iteration 2801 / 3675) loss: 1.345618
(Iteration 2901 / 3675) loss: 1.353949
(Epoch 12 / 15) train acc: 0.547000; val_acc: 0.513000
(Iteration 3001 / 3675) loss: 1.294126
(Iteration 3101 / 3675) loss: 1.318157
(Epoch 13 / 15) train acc: 0.547000; val_acc: 0.516000
(Iteration 3201 / 3675) loss: 1.371210
(Iteration 3301 / 3675) loss: 1.376482
(Iteration 3401 / 3675) loss: 1.352701
(Epoch 14 / 15) train acc: 0.510000; val_acc: 0.516000
(Iteration 3501 / 3675) loss: 1.262644
(Iteration 3601 / 3675) loss: 1.380328
(Epoch 15 / 15) train acc: 0.515000; val_acc: 0.522000
(Iteration 1 / 3675) loss: 2.302768
(Epoch 0 / 15) train acc: 0.120000; val_acc: 0.078000
(Iteration 101 / 3675) loss: 2.231682
(Iteration 201 / 3675) loss: 1.819599
(Epoch 1 / 15) train acc: 0.430000; val_acc: 0.394000
(Iteration 301 / 3675) loss: 1.488420
(Iteration 401 / 3675) loss: 1.542295
(Epoch 2 / 15) train acc: 0.501000; val_acc: 0.495000
(Iteration 501 / 3675) loss: 1.450119
(Iteration 601 / 3675) loss: 1.328020
(Iteration 701 / 3675) loss: 1.473561
(Epoch 3 / 15) train acc: 0.512000; val_acc: 0.519000
(Iteration 801 / 3675) loss: 1.382536
(Iteration 901 / 3675) loss: 1.430047
(Epoch 4 / 15) train acc: 0.546000; val_acc: 0.507000
(Iteration 1001 / 3675) loss: 1.421062
(Iteration 1101 / 3675) loss: 1.252944
(Iteration 1201 / 3675) loss: 1.317056
(Epoch 5 / 15) train acc: 0.520000; val_acc: 0.523000
(Iteration 1301 / 3675) loss: 1.395460
(Iteration 1401 / 3675) loss: 1.386850
(Epoch 6 / 15) train acc: 0.534000; val_acc: 0.530000
(Iteration 1501 / 3675) loss: 1.347884
(Iteration 1601 / 3675) loss: 1.336155
(Iteration 1701 / 3675) loss: 1.382672
(Epoch 7 / 15) train acc: 0.542000; val_acc: 0.535000
(Iteration 1801 / 3675) loss: 1.387346
(Iteration 1901 / 3675) loss: 1.255582
(Epoch 8 / 15) train acc: 0.553000; val_acc: 0.537000
(Iteration 2001 / 3675) loss: 1.318801
(Iteration 2101 / 3675) loss: 1.266030
(Iteration 2201 / 3675) loss: 1.295809
(Epoch 9 / 15) train acc: 0.537000; val_acc: 0.538000
```

```
(Iteration 2301 / 3675) loss: 1.375197
(Iteration 2401 / 3675) loss: 1.362714
(Epoch 10 / 15) train acc: 0.542000; val_acc: 0.536000
(Iteration 2501 / 3675) loss: 1.319703
(Iteration 2601 / 3675) loss: 1.389002
(Epoch 11 / 15) train acc: 0.621000; val_acc: 0.545000
(Iteration 2701 / 3675) loss: 1.324281
(Iteration 2801 / 3675) loss: 1.256157
(Iteration 2901 / 3675) loss: 1.394751
(Epoch 12 / 15) train acc: 0.563000; val_acc: 0.561000
(Iteration 3001 / 3675) loss: 1.288899
(Iteration 3101 / 3675) loss: 1.271087
(Epoch 13 / 15) train acc: 0.586000; val_acc: 0.560000
(Iteration 3201 / 3675) loss: 1.281904
(Iteration 3301 / 3675) loss: 1.299976
(Iteration 3401 / 3675) loss: 1.181749
(Epoch 14 / 15) train acc: 0.573000; val_acc: 0.565000
(Iteration 3501 / 3675) loss: 1.442868
(Iteration 3601 / 3675) loss: 1.279260
(Epoch 15 / 15) train acc: 0.609000; val_acc: 0.565000
(Iteration 1 / 3675) loss: 2.303009
(Epoch 0 / 15) train acc: 0.083000; val_acc: 0.112000
(Iteration 101 / 3675) loss: 2.244945
(Iteration 201 / 3675) loss: 1.853745
(Epoch 1 / 15) train acc: 0.412000; val_acc: 0.397000
(Iteration 301 / 3675) loss: 1.629943
(Iteration 401 / 3675) loss: 1.516430
(Epoch 2 / 15) train acc: 0.483000; val_acc: 0.493000
(Iteration 501 / 3675) loss: 1.495046
(Iteration 601 / 3675) loss: 1.474806
(Iteration 701 / 3675) loss: 1.461231
(Epoch 3 / 15) train acc: 0.537000; val_acc: 0.496000
(Iteration 801 / 3675) loss: 1.448488
(Iteration 901 / 3675) loss: 1.433521
(Epoch 4 / 15) train acc: 0.527000; val_acc: 0.521000
(Iteration 1001 / 3675) loss: 1.350580
(Iteration 1101 / 3675) loss: 1.549451
(Iteration 1201 / 3675) loss: 1.505640
(Epoch 5 / 15) train acc: 0.535000; val_acc: 0.533000
(Iteration 1301 / 3675) loss: 1.400468
(Iteration 1401 / 3675) loss: 1.465210
(Epoch 6 / 15) train acc: 0.537000; val_acc: 0.518000
(Iteration 1501 / 3675) loss: 1.415175
(Iteration 1601 / 3675) loss: 1.446363
(Iteration 1701 / 3675) loss: 1.380834
(Epoch 7 / 15) train acc: 0.535000; val_acc: 0.519000
(Iteration 1801 / 3675) loss: 1.495320
(Iteration 1901 / 3675) loss: 1.415364
```

```
(Epoch 8 / 15) train acc: 0.531000; val_acc: 0.519000
(Iteration 2001 / 3675) loss: 1.391444
(Iteration 2101 / 3675) loss: 1.445310
(Iteration 2201 / 3675) loss: 1.325114
(Epoch 9 / 15) train acc: 0.551000; val_acc: 0.529000
(Iteration 2301 / 3675) loss: 1.520084
(Iteration 2401 / 3675) loss: 1.363627
(Epoch 10 / 15) train acc: 0.529000; val_acc: 0.531000
(Iteration 2501 / 3675) loss: 1.400203
(Iteration 2601 / 3675) loss: 1.505067
(Epoch 11 / 15) train acc: 0.544000; val_acc: 0.535000
(Iteration 2701 / 3675) loss: 1.238790
(Iteration 2801 / 3675) loss: 1.433682
(Iteration 2901 / 3675) loss: 1.471810
(Epoch 12 / 15) train acc: 0.579000; val_acc: 0.538000
(Iteration 3001 / 3675) loss: 1.458025
(Iteration 3101 / 3675) loss: 1.405922
(Epoch 13 / 15) train acc: 0.593000; val_acc: 0.543000
(Iteration 3201 / 3675) loss: 1.408241
(Iteration 3301 / 3675) loss: 1.490388
(Iteration 3401 / 3675) loss: 1.428363
(Epoch 14 / 15) train acc: 0.573000; val_acc: 0.545000
(Iteration 3501 / 3675) loss: 1.394931
(Iteration 3601 / 3675) loss: 1.412484
(Epoch 15 / 15) train acc: 0.565000; val_acc: 0.551000
(Iteration 1 / 3675) loss: 2.302652
(Epoch 0 / 15) train acc: 0.096000; val_acc: 0.106000
(Iteration 101 / 3675) loss: 2.244100
(Iteration 201 / 3675) loss: 1.778786
(Epoch 1 / 15) train acc: 0.395000; val_acc: 0.403000
(Iteration 301 / 3675) loss: 1.574539
(Iteration 401 / 3675) loss: 1.438759
(Epoch 2 / 15) train acc: 0.523000; val_acc: 0.510000
(Iteration 501 / 3675) loss: 1.455284
(Iteration 601 / 3675) loss: 1.496589
(Iteration 701 / 3675) loss: 1.235914
(Epoch 3 / 15) train acc: 0.522000; val_acc: 0.521000
(Iteration 801 / 3675) loss: 1.288759
(Iteration 901 / 3675) loss: 1.234295
(Epoch 4 / 15) train acc: 0.528000; val_acc: 0.523000
(Iteration 1001 / 3675) loss: 1.339054
(Iteration 1101 / 3675) loss: 1.470111
(Iteration 1201 / 3675) loss: 1.252266
(Epoch 5 / 15) train acc: 0.540000; val_acc: 0.530000
(Iteration 1301 / 3675) loss: 1.225571
(Iteration 1401 / 3675) loss: 1.243373
(Epoch 6 / 15) train acc: 0.547000; val_acc: 0.530000
(Iteration 1501 / 3675) loss: 1.461758
```

```
(Iteration 1601 / 3675) loss: 1.407997
(Iteration 1701 / 3675) loss: 1.270701
(Epoch 7 / 15) train acc: 0.583000; val_acc: 0.535000
(Iteration 1801 / 3675) loss: 1.317456
(Iteration 1901 / 3675) loss: 1.302431
(Epoch 8 / 15) train acc: 0.563000; val_acc: 0.548000
(Iteration 2001 / 3675) loss: 1.163732
(Iteration 2101 / 3675) loss: 1.273827
(Iteration 2201 / 3675) loss: 1.183162
(Epoch 9 / 15) train acc: 0.575000; val_acc: 0.559000
(Iteration 2301 / 3675) loss: 1.180357
(Iteration 2401 / 3675) loss: 1.192630
(Epoch 10 / 15) train acc: 0.566000; val_acc: 0.559000
(Iteration 2501 / 3675) loss: 1.212690
(Iteration 2601 / 3675) loss: 1.078855
(Epoch 11 / 15) train acc: 0.595000; val_acc: 0.563000
(Iteration 2701 / 3675) loss: 1.184676
(Iteration 2801 / 3675) loss: 1.238743
(Iteration 2901 / 3675) loss: 1.106662
(Epoch 12 / 15) train acc: 0.612000; val_acc: 0.562000
(Iteration 3001 / 3675) loss: 1.114469
(Iteration 3101 / 3675) loss: 1.196679
(Epoch 13 / 15) train acc: 0.605000; val_acc: 0.567000
(Iteration 3201 / 3675) loss: 1.109021
(Iteration 3301 / 3675) loss: 1.206731
(Iteration 3401 / 3675) loss: 1.240773
(Epoch 14 / 15) train acc: 0.636000; val_acc: 0.571000
(Iteration 3501 / 3675) loss: 1.137610
(Iteration 3601 / 3675) loss: 1.130964
(Epoch 15 / 15) train acc: 0.624000; val_acc: 0.574000
lr 1.000000e-02 reg 1.000000e-03 dim 5.000000e+02 train accuracy: 0.426000 val
accuracy: 0.415000
lr 1.000000e-02 reg 5.000000e-03 dim 5.000000e+02 train accuracy: 0.398000 val
accuracy: 0.414000
lr 1.000000e-02 reg 1.000000e-02 dim 5.000000e+02 train accuracy: 0.432000 val
accuracy: 0.409000
lr 3.000000e-02 reg 1.000000e-03 dim 5.000000e+02 train accuracy: 0.515000 val
accuracy: 0.522000
lr 3.000000e-02 reg 5.000000e-03 dim 5.000000e+02 train accuracy: 0.524000 val
accuracy: 0.525000
lr 3.000000e-02 reg 1.000000e-02 dim 5.000000e+02 train accuracy: 0.534000 val
accuracy: 0.521000
lr 1.000000e-01 reg 1.000000e-03 dim 5.000000e+02 train accuracy: 0.624000 val
accuracy: 0.574000
lr 1.000000e-01 reg 5.000000e-03 dim 5.000000e+02 train accuracy: 0.609000 val
accuracy: 0.565000
lr 1.000000e-01 reg 1.000000e-02 dim 5.000000e+02 train accuracy: 0.565000 val
accuracy: 0.551000
```

```
best validation accuracy achieved: 0.574000
```

[45]:
```python
# Run your best neural net classifier on the test set. You should be able
# to get more than 55% accuracy.

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

```
0.564
```

FullyConnectedNets

January 31, 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 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.

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

[4]:
```
# 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.

[4]:
```
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.

[5]:
```
# 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-3,1e-2,1e-1]  # Experiment with this!
learning_rate = [1e-5,1e-4,1e-3,1e-2]  # Experiment with this!
best_net = None
results = {}
best_train = 0

for i in weight_scale:
  for j in learning_rate:
    model = FullyConnectedNet(
        [100, 100],
        weight_scale=i,
        dtype=np.float64
    )
    solver = Solver(
        model,
        small_data,
        print_every=10,
        num_epochs=20,
        batch_size=25,
        update_rule="sgd",
        optim_config={"learning_rate": j},
    )
    solver.train()

    if solver.train_acc_history[len(solver.train_acc_history) - 1] > best_train:
        best_net = model
        best_solver = solver
        best_train = solver.train_acc_history[len(solver.train_acc_history) - 1]

    results[(i,j)] = (solver.train_acc_history[len(solver.train_acc_history) -␣
 ↪1], solver.best_val_acc)

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

print('best train accuracy achieved: %f' % best_train)

plt.plot(best_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: 2.302510
(Epoch 0 / 20) train acc: 0.120000; val_acc: 0.105000
(Epoch 1 / 20) train acc: 0.120000; val_acc: 0.107000
(Epoch 2 / 20) train acc: 0.120000; val_acc: 0.107000
(Epoch 3 / 20) train acc: 0.120000; val_acc: 0.108000
(Epoch 4 / 20) train acc: 0.120000; val_acc: 0.109000
(Epoch 5 / 20) train acc: 0.120000; val_acc: 0.109000
(Iteration 11 / 40) loss: 2.302563
(Epoch 6 / 20) train acc: 0.120000; val_acc: 0.108000
(Epoch 7 / 20) train acc: 0.120000; val_acc: 0.108000
(Epoch 8 / 20) train acc: 0.120000; val_acc: 0.108000
(Epoch 9 / 20) train acc: 0.120000; val_acc: 0.110000
(Epoch 10 / 20) train acc: 0.120000; val_acc: 0.110000
(Iteration 21 / 40) loss: 2.302531
(Epoch 11 / 20) train acc: 0.120000; val_acc: 0.109000
(Epoch 12 / 20) train acc: 0.120000; val_acc: 0.109000
(Epoch 13 / 20) train acc: 0.120000; val_acc: 0.109000
(Epoch 14 / 20) train acc: 0.120000; val_acc: 0.109000
(Epoch 15 / 20) train acc: 0.120000; val_acc: 0.108000
(Iteration 31 / 40) loss: 2.302554
(Epoch 16 / 20) train acc: 0.120000; val_acc: 0.108000
(Epoch 17 / 20) train acc: 0.120000; val_acc: 0.110000
(Epoch 18 / 20) train acc: 0.120000; val_acc: 0.107000
(Epoch 19 / 20) train acc: 0.120000; val_acc: 0.107000
(Epoch 20 / 20) train acc: 0.120000; val_acc: 0.107000
(Iteration 1 / 40) loss: 2.302594
(Epoch 0 / 20) train acc: 0.140000; val_acc: 0.083000
(Epoch 1 / 20) train acc: 0.160000; val_acc: 0.086000
(Epoch 2 / 20) train acc: 0.160000; val_acc: 0.084000
(Epoch 3 / 20) train acc: 0.160000; val_acc: 0.081000
(Epoch 4 / 20) train acc: 0.160000; val_acc: 0.076000
(Epoch 5 / 20) train acc: 0.160000; val_acc: 0.081000
(Iteration 11 / 40) loss: 2.302568
(Epoch 6 / 20) train acc: 0.160000; val_acc: 0.080000
(Epoch 7 / 20) train acc: 0.160000; val_acc: 0.083000
(Epoch 8 / 20) train acc: 0.160000; val_acc: 0.083000
(Epoch 9 / 20) train acc: 0.160000; val_acc: 0.082000
(Epoch 10 / 20) train acc: 0.160000; val_acc: 0.083000
(Iteration 21 / 40) loss: 2.302517
(Epoch 11 / 20) train acc: 0.160000; val_acc: 0.082000
(Epoch 12 / 20) train acc: 0.160000; val_acc: 0.081000
```

```
(Epoch 13 / 20) train acc: 0.160000; val_acc: 0.081000
(Epoch 14 / 20) train acc: 0.160000; val_acc: 0.081000
(Epoch 15 / 20) train acc: 0.160000; val_acc: 0.081000
(Iteration 31 / 40) loss: 2.302535
(Epoch 16 / 20) train acc: 0.160000; val_acc: 0.081000
(Epoch 17 / 20) train acc: 0.160000; val_acc: 0.081000
(Epoch 18 / 20) train acc: 0.160000; val_acc: 0.081000
(Epoch 19 / 20) train acc: 0.160000; val_acc: 0.080000
(Epoch 20 / 20) train acc: 0.160000; val_acc: 0.081000
(Iteration 1 / 40) loss: 2.302533
(Epoch 0 / 20) train acc: 0.160000; val_acc: 0.093000
(Epoch 1 / 20) train acc: 0.100000; val_acc: 0.116000
(Epoch 2 / 20) train acc: 0.180000; val_acc: 0.116000
(Epoch 3 / 20) train acc: 0.180000; val_acc: 0.112000
(Epoch 4 / 20) train acc: 0.160000; val_acc: 0.112000
(Epoch 5 / 20) train acc: 0.160000; val_acc: 0.113000
(Iteration 11 / 40) loss: 2.302348
(Epoch 6 / 20) train acc: 0.160000; val_acc: 0.113000
(Epoch 7 / 20) train acc: 0.160000; val_acc: 0.113000
(Epoch 8 / 20) train acc: 0.160000; val_acc: 0.112000
(Epoch 9 / 20) train acc: 0.160000; val_acc: 0.112000
(Epoch 10 / 20) train acc: 0.160000; val_acc: 0.112000
(Iteration 21 / 40) loss: 2.302099
(Epoch 11 / 20) train acc: 0.160000; val_acc: 0.112000
(Epoch 12 / 20) train acc: 0.160000; val_acc: 0.112000
(Epoch 13 / 20) train acc: 0.160000; val_acc: 0.112000
(Epoch 14 / 20) train acc: 0.160000; val_acc: 0.112000
(Epoch 15 / 20) train acc: 0.160000; val_acc: 0.112000
(Iteration 31 / 40) loss: 2.302085
(Epoch 16 / 20) train acc: 0.160000; val_acc: 0.112000
(Epoch 17 / 20) train acc: 0.160000; val_acc: 0.112000
(Epoch 18 / 20) train acc: 0.160000; val_acc: 0.112000
(Epoch 19 / 20) train acc: 0.160000; val_acc: 0.112000
(Epoch 20 / 20) train acc: 0.160000; val_acc: 0.112000
(Iteration 1 / 40) loss: 2.302667
(Epoch 0 / 20) train acc: 0.140000; val_acc: 0.104000
(Epoch 1 / 20) train acc: 0.160000; val_acc: 0.079000
(Epoch 2 / 20) train acc: 0.180000; val_acc: 0.101000
(Epoch 3 / 20) train acc: 0.180000; val_acc: 0.106000
(Epoch 4 / 20) train acc: 0.160000; val_acc: 0.079000
(Epoch 5 / 20) train acc: 0.160000; val_acc: 0.079000
(Iteration 11 / 40) loss: 2.300575
(Epoch 6 / 20) train acc: 0.160000; val_acc: 0.079000
(Epoch 7 / 20) train acc: 0.160000; val_acc: 0.079000
(Epoch 8 / 20) train acc: 0.160000; val_acc: 0.079000
(Epoch 9 / 20) train acc: 0.160000; val_acc: 0.079000
(Epoch 10 / 20) train acc: 0.160000; val_acc: 0.079000
(Iteration 21 / 40) loss: 2.297936
```

```
(Epoch 11 / 20) train acc: 0.160000; val_acc: 0.079000
(Epoch 12 / 20) train acc: 0.160000; val_acc: 0.079000
(Epoch 13 / 20) train acc: 0.160000; val_acc: 0.079000
(Epoch 14 / 20) train acc: 0.160000; val_acc: 0.079000
(Epoch 15 / 20) train acc: 0.160000; val_acc: 0.079000
(Iteration 31 / 40) loss: 2.296367
(Epoch 16 / 20) train acc: 0.160000; val_acc: 0.079000
(Epoch 17 / 20) train acc: 0.160000; val_acc: 0.079000
(Epoch 18 / 20) train acc: 0.160000; val_acc: 0.079000
(Epoch 19 / 20) train acc: 0.160000; val_acc: 0.079000
(Epoch 20 / 20) train acc: 0.160000; val_acc: 0.079000
(Iteration 1 / 40) loss: 2.282399
(Epoch 0 / 20) train acc: 0.040000; val_acc: 0.084000
(Epoch 1 / 20) train acc: 0.040000; val_acc: 0.084000
(Epoch 2 / 20) train acc: 0.040000; val_acc: 0.084000
(Epoch 3 / 20) train acc: 0.040000; val_acc: 0.084000
(Epoch 4 / 20) train acc: 0.040000; val_acc: 0.084000
(Epoch 5 / 20) train acc: 0.040000; val_acc: 0.085000
(Iteration 11 / 40) loss: 2.346485
(Epoch 6 / 20) train acc: 0.040000; val_acc: 0.085000
(Epoch 7 / 20) train acc: 0.040000; val_acc: 0.085000
(Epoch 8 / 20) train acc: 0.040000; val_acc: 0.085000
(Epoch 9 / 20) train acc: 0.040000; val_acc: 0.085000
(Epoch 10 / 20) train acc: 0.060000; val_acc: 0.085000
(Iteration 21 / 40) loss: 2.394382
(Epoch 11 / 20) train acc: 0.060000; val_acc: 0.085000
(Epoch 12 / 20) train acc: 0.060000; val_acc: 0.085000
(Epoch 13 / 20) train acc: 0.060000; val_acc: 0.085000
(Epoch 14 / 20) train acc: 0.060000; val_acc: 0.085000
(Epoch 15 / 20) train acc: 0.060000; val_acc: 0.086000
(Iteration 31 / 40) loss: 2.312257
(Epoch 16 / 20) train acc: 0.060000; val_acc: 0.086000
(Epoch 17 / 20) train acc: 0.060000; val_acc: 0.086000
(Epoch 18 / 20) train acc: 0.060000; val_acc: 0.086000
(Epoch 19 / 20) train acc: 0.060000; val_acc: 0.087000
(Epoch 20 / 20) train acc: 0.060000; val_acc: 0.086000
(Iteration 1 / 40) loss: 2.338080
(Epoch 0 / 20) train acc: 0.020000; val_acc: 0.089000
(Epoch 1 / 20) train acc: 0.020000; val_acc: 0.088000
(Epoch 2 / 20) train acc: 0.020000; val_acc: 0.086000
(Epoch 3 / 20) train acc: 0.020000; val_acc: 0.084000
(Epoch 4 / 20) train acc: 0.020000; val_acc: 0.086000
(Epoch 5 / 20) train acc: 0.020000; val_acc: 0.086000
(Iteration 11 / 40) loss: 2.301028
(Epoch 6 / 20) train acc: 0.020000; val_acc: 0.086000
(Epoch 7 / 20) train acc: 0.040000; val_acc: 0.086000
(Epoch 8 / 20) train acc: 0.040000; val_acc: 0.089000
(Epoch 9 / 20) train acc: 0.040000; val_acc: 0.095000
```

```
(Epoch 10 / 20) train acc: 0.040000; val_acc: 0.097000
(Iteration 21 / 40) loss: 2.292944
(Epoch 11 / 20) train acc: 0.040000; val_acc: 0.096000
(Epoch 12 / 20) train acc: 0.040000; val_acc: 0.097000
(Epoch 13 / 20) train acc: 0.040000; val_acc: 0.094000
(Epoch 14 / 20) train acc: 0.060000; val_acc: 0.093000
(Epoch 15 / 20) train acc: 0.060000; val_acc: 0.095000
(Iteration 31 / 40) loss: 2.330349
(Epoch 16 / 20) train acc: 0.060000; val_acc: 0.096000
(Epoch 17 / 20) train acc: 0.080000; val_acc: 0.097000
(Epoch 18 / 20) train acc: 0.080000; val_acc: 0.098000
(Epoch 19 / 20) train acc: 0.080000; val_acc: 0.096000
(Epoch 20 / 20) train acc: 0.100000; val_acc: 0.094000
(Iteration 1 / 40) loss: 2.408740
(Epoch 0 / 20) train acc: 0.100000; val_acc: 0.129000
(Epoch 1 / 20) train acc: 0.160000; val_acc: 0.129000
(Epoch 2 / 20) train acc: 0.200000; val_acc: 0.129000
(Epoch 3 / 20) train acc: 0.220000; val_acc: 0.136000
(Epoch 4 / 20) train acc: 0.260000; val_acc: 0.140000
(Epoch 5 / 20) train acc: 0.380000; val_acc: 0.142000
(Iteration 11 / 40) loss: 2.159415
(Epoch 6 / 20) train acc: 0.360000; val_acc: 0.145000
(Epoch 7 / 20) train acc: 0.400000; val_acc: 0.149000
(Epoch 8 / 20) train acc: 0.400000; val_acc: 0.146000
(Epoch 9 / 20) train acc: 0.400000; val_acc: 0.153000
(Epoch 10 / 20) train acc: 0.400000; val_acc: 0.154000
(Iteration 21 / 40) loss: 1.957353
(Epoch 11 / 20) train acc: 0.420000; val_acc: 0.144000
(Epoch 12 / 20) train acc: 0.400000; val_acc: 0.148000
(Epoch 13 / 20) train acc: 0.440000; val_acc: 0.165000
(Epoch 14 / 20) train acc: 0.480000; val_acc: 0.164000
(Epoch 15 / 20) train acc: 0.440000; val_acc: 0.154000
(Iteration 31 / 40) loss: 1.957044
(Epoch 16 / 20) train acc: 0.460000; val_acc: 0.158000
(Epoch 17 / 20) train acc: 0.440000; val_acc: 0.158000
(Epoch 18 / 20) train acc: 0.440000; val_acc: 0.161000
(Epoch 19 / 20) train acc: 0.440000; val_acc: 0.165000
(Epoch 20 / 20) train acc: 0.480000; val_acc: 0.170000
(Iteration 1 / 40) loss: 2.233375
(Epoch 0 / 20) train acc: 0.220000; val_acc: 0.113000
(Epoch 1 / 20) train acc: 0.260000; val_acc: 0.093000
(Epoch 2 / 20) train acc: 0.500000; val_acc: 0.138000
(Epoch 3 / 20) train acc: 0.480000; val_acc: 0.147000
(Epoch 4 / 20) train acc: 0.400000; val_acc: 0.149000
(Epoch 5 / 20) train acc: 0.800000; val_acc: 0.170000
(Iteration 11 / 40) loss: 0.981903
(Epoch 6 / 20) train acc: 0.840000; val_acc: 0.176000
(Epoch 7 / 20) train acc: 0.860000; val_acc: 0.173000
```

```
(Epoch 8 / 20) train acc: 0.860000; val_acc: 0.181000
(Epoch 9 / 20) train acc: 0.940000; val_acc: 0.173000
(Epoch 10 / 20) train acc: 0.940000; val_acc: 0.189000
(Iteration 21 / 40) loss: 0.265250
(Epoch 11 / 20) train acc: 0.960000; val_acc: 0.180000
(Epoch 12 / 20) train acc: 0.900000; val_acc: 0.165000
(Epoch 13 / 20) train acc: 0.960000; val_acc: 0.194000
(Epoch 14 / 20) train acc: 0.980000; val_acc: 0.191000
(Epoch 15 / 20) train acc: 0.960000; val_acc: 0.193000
(Iteration 31 / 40) loss: 0.071444
(Epoch 16 / 20) train acc: 0.980000; val_acc: 0.190000
(Epoch 17 / 20) train acc: 1.000000; val_acc: 0.192000
(Epoch 18 / 20) train acc: 0.980000; val_acc: 0.183000
(Epoch 19 / 20) train acc: 0.980000; val_acc: 0.187000
(Epoch 20 / 20) train acc: 1.000000; val_acc: 0.183000
(Iteration 1 / 40) loss: inf
```

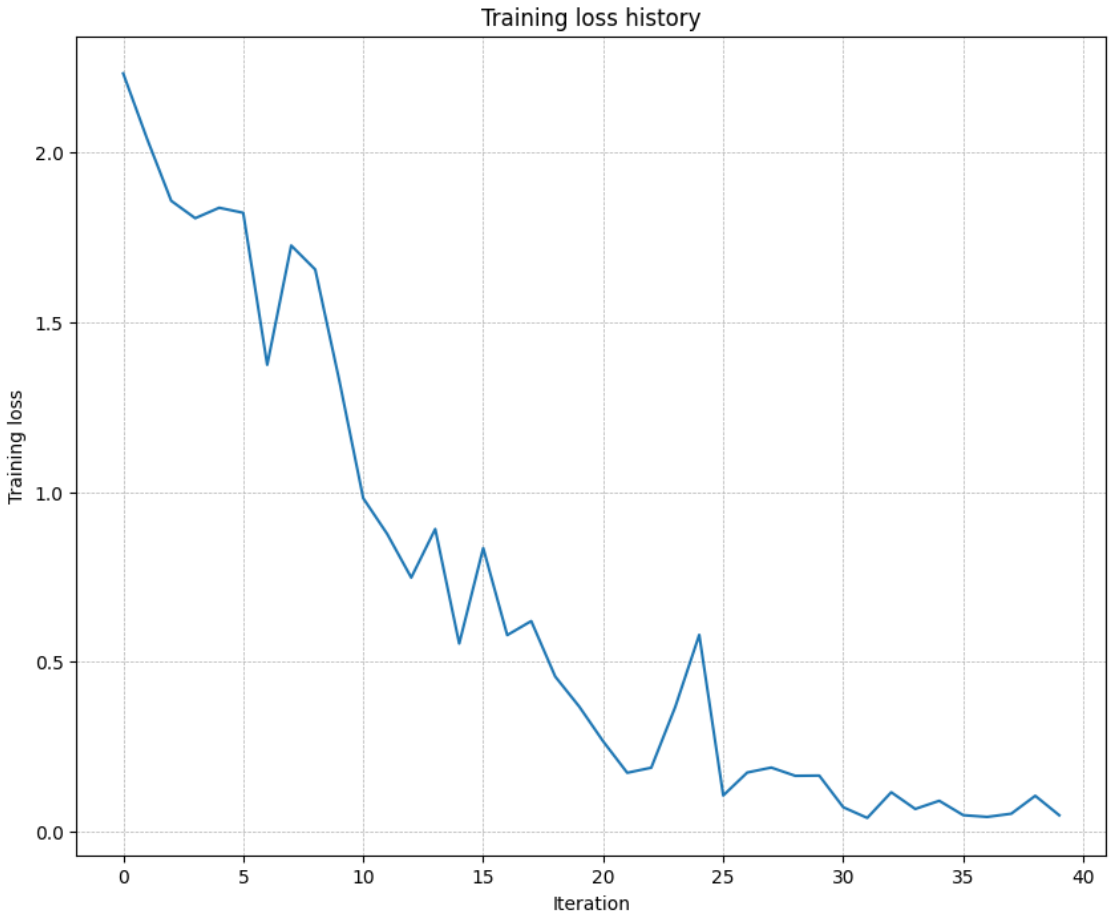
```
/content/drive/My Drive/cse493g1/assignments/assignment2/cse493g1/layers.py:205:
RuntimeWarning: overflow encountered in exp
  x = np.exp(x)
/content/drive/My Drive/cse493g1/assignments/assignment2/cse493g1/layers.py:207:
RuntimeWarning: invalid value encountered in divide
  x = x / row_sums[:,np.newaxis]
/content/drive/My Drive/cse493g1/assignments/assignment2/cse493g1/layers.py:210:
RuntimeWarning: divide by zero encountered in log
  loss = np.sum(-np.log(loss_array))
```

```
(Epoch 0 / 20) train acc: 0.080000; val_acc: 0.087000
(Epoch 1 / 20) train acc: 0.080000; val_acc: 0.087000
(Epoch 2 / 20) train acc: 0.080000; val_acc: 0.087000
(Epoch 3 / 20) train acc: 0.080000; val_acc: 0.087000
(Epoch 4 / 20) train acc: 0.080000; val_acc: 0.087000
(Epoch 5 / 20) train acc: 0.080000; val_acc: 0.087000
(Iteration 11 / 40) loss: nan
(Epoch 6 / 20) train acc: 0.080000; val_acc: 0.087000
(Epoch 7 / 20) train acc: 0.080000; val_acc: 0.087000
(Epoch 8 / 20) train acc: 0.080000; val_acc: 0.087000
(Epoch 9 / 20) train acc: 0.080000; val_acc: 0.087000
(Epoch 10 / 20) train acc: 0.080000; val_acc: 0.087000
(Iteration 21 / 40) loss: nan
(Epoch 11 / 20) train acc: 0.080000; val_acc: 0.087000
(Epoch 12 / 20) train acc: 0.080000; val_acc: 0.087000
(Epoch 13 / 20) train acc: 0.080000; val_acc: 0.087000
(Epoch 14 / 20) train acc: 0.080000; val_acc: 0.087000
(Epoch 15 / 20) train acc: 0.080000; val_acc: 0.087000
(Iteration 31 / 40) loss: nan
(Epoch 16 / 20) train acc: 0.080000; val_acc: 0.087000
(Epoch 17 / 20) train acc: 0.080000; val_acc: 0.087000
(Epoch 18 / 20) train acc: 0.080000; val_acc: 0.087000
```

```
(Epoch 19 / 20) train acc: 0.080000; val_acc: 0.087000
(Epoch 20 / 20) train acc: 0.080000; val_acc: 0.087000
(Iteration 1 / 40) loss: 175.555031
(Epoch 0 / 20) train acc: 0.140000; val_acc: 0.089000
(Epoch 1 / 20) train acc: 0.180000; val_acc: 0.099000
(Epoch 2 / 20) train acc: 0.300000; val_acc: 0.097000
(Epoch 3 / 20) train acc: 0.400000; val_acc: 0.101000
(Epoch 4 / 20) train acc: 0.340000; val_acc: 0.110000
(Epoch 5 / 20) train acc: 0.540000; val_acc: 0.108000
(Iteration 11 / 40) loss: 37.685187
(Epoch 6 / 20) train acc: 0.600000; val_acc: 0.100000
(Epoch 7 / 20) train acc: 0.700000; val_acc: 0.102000
(Epoch 8 / 20) train acc: 0.800000; val_acc: 0.106000
(Epoch 9 / 20) train acc: 0.760000; val_acc: 0.107000
(Epoch 10 / 20) train acc: 0.740000; val_acc: 0.105000
(Iteration 21 / 40) loss: 9.206001
(Epoch 11 / 20) train acc: 0.800000; val_acc: 0.101000
(Epoch 12 / 20) train acc: 0.900000; val_acc: 0.104000
(Epoch 13 / 20) train acc: 0.940000; val_acc: 0.106000
(Epoch 14 / 20) train acc: 0.900000; val_acc: 0.104000
(Epoch 15 / 20) train acc: 0.920000; val_acc: 0.102000
(Iteration 31 / 40) loss: 23.937100
(Epoch 16 / 20) train acc: 0.940000; val_acc: 0.106000
(Epoch 17 / 20) train acc: 0.920000; val_acc: 0.105000
(Epoch 18 / 20) train acc: 0.880000; val_acc: 0.103000
(Epoch 19 / 20) train acc: 0.920000; val_acc: 0.100000
(Epoch 20 / 20) train acc: 0.960000; val_acc: 0.103000
(Iteration 1 / 40) loss: 232.212479
(Epoch 0 / 20) train acc: 0.260000; val_acc: 0.108000
(Epoch 1 / 20) train acc: 0.080000; val_acc: 0.087000
(Epoch 2 / 20) train acc: 0.080000; val_acc: 0.087000
(Epoch 3 / 20) train acc: 0.080000; val_acc: 0.087000
(Epoch 4 / 20) train acc: 0.080000; val_acc: 0.087000
(Epoch 5 / 20) train acc: 0.080000; val_acc: 0.087000
(Iteration 11 / 40) loss: nan
(Epoch 6 / 20) train acc: 0.080000; val_acc: 0.087000
(Epoch 7 / 20) train acc: 0.080000; val_acc: 0.087000
(Epoch 8 / 20) train acc: 0.080000; val_acc: 0.087000
(Epoch 9 / 20) train acc: 0.080000; val_acc: 0.087000
(Epoch 10 / 20) train acc: 0.080000; val_acc: 0.087000
(Iteration 21 / 40) loss: nan
(Epoch 11 / 20) train acc: 0.080000; val_acc: 0.087000
(Epoch 12 / 20) train acc: 0.080000; val_acc: 0.087000
(Epoch 13 / 20) train acc: 0.080000; val_acc: 0.087000
(Epoch 14 / 20) train acc: 0.080000; val_acc: 0.087000
(Epoch 15 / 20) train acc: 0.080000; val_acc: 0.087000
(Iteration 31 / 40) loss: nan
(Epoch 16 / 20) train acc: 0.080000; val_acc: 0.087000
```

```
(Epoch 17 / 20) train acc: 0.080000; val_acc: 0.087000
(Epoch 18 / 20) train acc: 0.080000; val_acc: 0.087000
(Epoch 19 / 20) train acc: 0.080000; val_acc: 0.087000
(Epoch 20 / 20) train acc: 0.080000; val_acc: 0.087000
(Iteration 1 / 40) loss: inf
(Epoch 0 / 20) train acc: 0.200000; val_acc: 0.090000
(Epoch 1 / 20) train acc: 0.080000; val_acc: 0.087000
(Epoch 2 / 20) train acc: 0.080000; val_acc: 0.087000
(Epoch 3 / 20) train acc: 0.080000; val_acc: 0.087000
(Epoch 4 / 20) train acc: 0.080000; val_acc: 0.087000
(Epoch 5 / 20) train acc: 0.080000; val_acc: 0.087000
(Iteration 11 / 40) loss: nan
(Epoch 6 / 20) train acc: 0.080000; val_acc: 0.087000
(Epoch 7 / 20) train acc: 0.080000; val_acc: 0.087000
(Epoch 8 / 20) train acc: 0.080000; val_acc: 0.087000
(Epoch 9 / 20) train acc: 0.080000; val_acc: 0.087000
(Epoch 10 / 20) train acc: 0.080000; val_acc: 0.087000
(Iteration 21 / 40) loss: nan
(Epoch 11 / 20) train acc: 0.080000; val_acc: 0.087000
(Epoch 12 / 20) train acc: 0.080000; val_acc: 0.087000
(Epoch 13 / 20) train acc: 0.080000; val_acc: 0.087000
(Epoch 14 / 20) train acc: 0.080000; val_acc: 0.087000
(Epoch 15 / 20) train acc: 0.080000; val_acc: 0.087000
(Iteration 31 / 40) loss: nan
(Epoch 16 / 20) train acc: 0.080000; val_acc: 0.087000
(Epoch 17 / 20) train acc: 0.080000; val_acc: 0.087000
(Epoch 18 / 20) train acc: 0.080000; val_acc: 0.087000
(Epoch 19 / 20) train acc: 0.080000; val_acc: 0.087000
(Epoch 20 / 20) train acc: 0.080000; val_acc: 0.087000
weight_scale  1.000000e-03 lr 1.000000e-05 train accuracy: 0.120000 val
accuracy: 0.110000
weight_scale  1.000000e-03 lr 1.000000e-04 train accuracy: 0.160000 val
accuracy: 0.086000
weight_scale  1.000000e-03 lr 1.000000e-03 train accuracy: 0.160000 val
accuracy: 0.116000
weight_scale  1.000000e-03 lr 1.000000e-02 train accuracy: 0.160000 val
accuracy: 0.106000
weight_scale  1.000000e-02 lr 1.000000e-05 train accuracy: 0.060000 val
accuracy: 0.087000
weight_scale  1.000000e-02 lr 1.000000e-04 train accuracy: 0.100000 val
accuracy: 0.098000
weight_scale  1.000000e-02 lr 1.000000e-03 train accuracy: 0.480000 val
accuracy: 0.170000
weight_scale  1.000000e-02 lr 1.000000e-02 train accuracy: 1.000000 val
accuracy: 0.194000
weight_scale  1.000000e-01 lr 1.000000e-05 train accuracy: 0.080000 val
accuracy: 0.087000
weight_scale  1.000000e-01 lr 1.000000e-04 train accuracy: 0.960000 val
```

```
accuracy: 0.110000
weight_scale  1.000000e-01 lr 1.000000e-03 train accuracy: 0.080000 val
accuracy: 0.108000
weight_scale  1.000000e-01 lr 1.000000e-02 train accuracy: 0.080000 val
accuracy: 0.090000
best train accuracy achieved: 1.000000
```



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.

[8]:
```
# 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'],
```

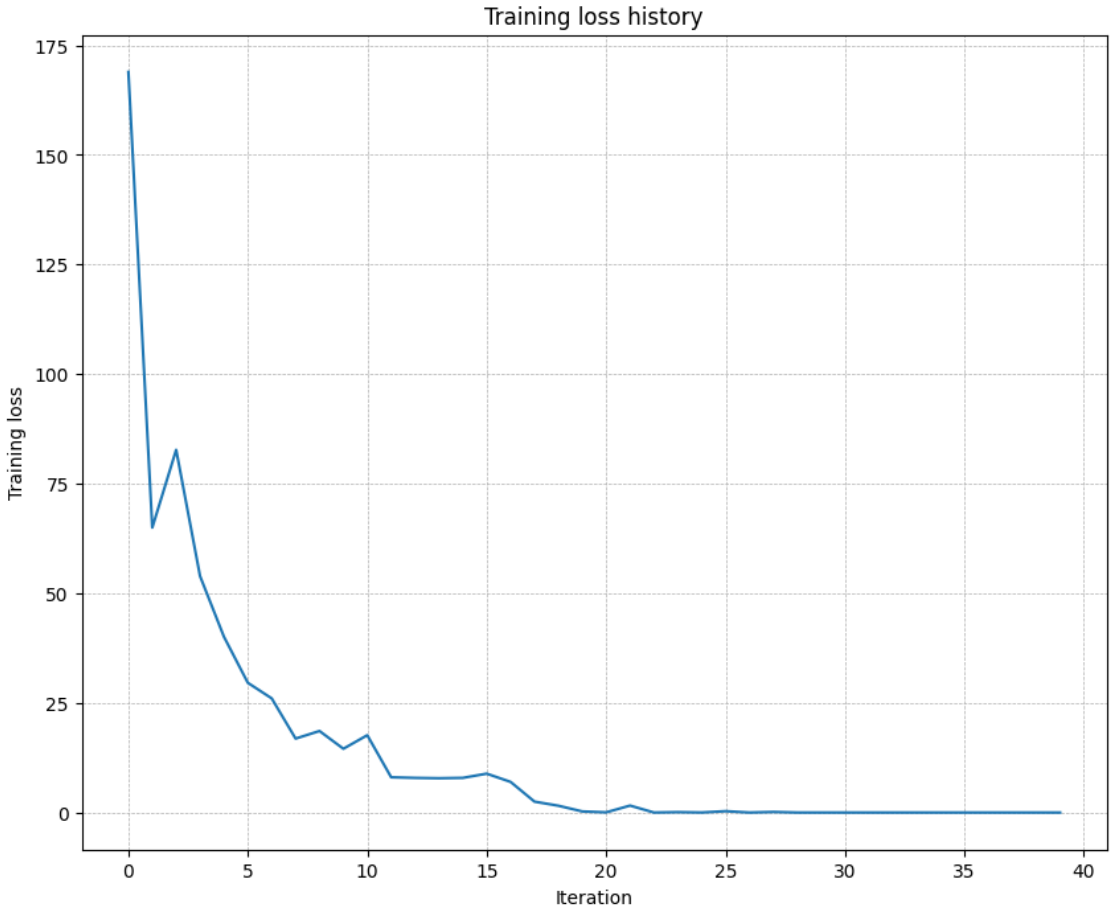
```
  'y_val': data['y_val'],
}


weight_scale = [1e-5,1e-4,1e-3,1e-2,1e-1]  # Experiment with this!
learning_rate = [2e-5,2e-4,2e-3,2e-2,2e-1]  # Experiment with this!
best_net = None
results = {}
best_train = 0

for i in weight_scale:
  for j in learning_rate:
    model = FullyConnectedNet(
        [100, 100, 100, 100],
        weight_scale = i,
        dtype=np.float64
    )
    solver = Solver(
        model,
        small_data,
        print_every=10,
        num_epochs=20,
        batch_size=25,
        update_rule="sgd",
        optim_config={"learning_rate": j},
        verbose = False,
    )
    solver.train()

    if solver.train_acc_history[len(solver.train_acc_history) - 1] > best_train:
        best_net = model
        best_solver = solver
        best_train = solver.train_acc_history[len(solver.train_acc_history) - 1]

    results[(i,j)] = (solver.train_acc_history[len(solver.train_acc_history) -␣
 ↪1], solver.best_val_acc)

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

print('best train accuracy achieved: %f' % best_train)

plt.plot(best_solver.loss_history)
plt.title('Training loss history')
```

```
plt.xlabel('Iteration')
plt.ylabel('Training loss')
plt.grid(linestyle='--', linewidth=0.5)
plt.show()
```

```
/content/drive/My Drive/cse493g1/assignments/assignment2/cse493g1/layers.py:210:
RuntimeWarning: divide by zero encountered in log
  loss = np.sum(-np.log(loss_array))
/content/drive/My Drive/cse493g1/assignments/assignment2/cse493g1/layers.py:205:
RuntimeWarning: overflow encountered in exp
  x = np.exp(x)
/content/drive/My Drive/cse493g1/assignments/assignment2/cse493g1/layers.py:207:
RuntimeWarning: invalid value encountered in divide
  x = x / row_sums[:,np.newaxis]
```

```
weight_scale  1.000000e-05 lr 2.000000e-05 train accuracy: 0.160000 val
accuracy: 0.112000
weight_scale  1.000000e-05 lr 2.000000e-04 train accuracy: 0.160000 val
accuracy: 0.119000
weight_scale  1.000000e-05 lr 2.000000e-03 train accuracy: 0.160000 val
accuracy: 0.119000
weight_scale  1.000000e-05 lr 2.000000e-02 train accuracy: 0.160000 val
accuracy: 0.119000
weight_scale  1.000000e-05 lr 2.000000e-01 train accuracy: 0.160000 val
accuracy: 0.112000
weight_scale  1.000000e-04 lr 2.000000e-05 train accuracy: 0.160000 val
accuracy: 0.112000
weight_scale  1.000000e-04 lr 2.000000e-04 train accuracy: 0.160000 val
accuracy: 0.112000
weight_scale  1.000000e-04 lr 2.000000e-03 train accuracy: 0.160000 val
accuracy: 0.079000
weight_scale  1.000000e-04 lr 2.000000e-02 train accuracy: 0.160000 val
accuracy: 0.119000
weight_scale  1.000000e-04 lr 2.000000e-01 train accuracy: 0.160000 val
accuracy: 0.079000
weight_scale  1.000000e-03 lr 2.000000e-05 train accuracy: 0.160000 val
accuracy: 0.112000
weight_scale  1.000000e-03 lr 2.000000e-04 train accuracy: 0.160000 val
accuracy: 0.108000
weight_scale  1.000000e-03 lr 2.000000e-03 train accuracy: 0.160000 val
accuracy: 0.112000
weight_scale  1.000000e-03 lr 2.000000e-02 train accuracy: 0.160000 val
accuracy: 0.119000
weight_scale  1.000000e-03 lr 2.000000e-01 train accuracy: 0.160000 val
accuracy: 0.112000
weight_scale  1.000000e-02 lr 2.000000e-05 train accuracy: 0.140000 val
accuracy: 0.090000
weight_scale  1.000000e-02 lr 2.000000e-04 train accuracy: 0.100000 val
```

```
accuracy: 0.107000
weight_scale  1.000000e-02 lr 2.000000e-03 train accuracy: 0.160000 val
accuracy: 0.113000
weight_scale  1.000000e-02 lr 2.000000e-02 train accuracy: 0.160000 val
accuracy: 0.079000
weight_scale  1.000000e-02 lr 2.000000e-01 train accuracy: 0.240000 val
accuracy: 0.141000
weight_scale  1.000000e-01 lr 2.000000e-05 train accuracy: 0.340000 val
accuracy: 0.114000
weight_scale  1.000000e-01 lr 2.000000e-04 train accuracy: 1.000000 val
accuracy: 0.129000
weight_scale  1.000000e-01 lr 2.000000e-03 train accuracy: 1.000000 val
accuracy: 0.136000
weight_scale  1.000000e-01 lr 2.000000e-02 train accuracy: 0.080000 val
accuracy: 0.112000
weight_scale  1.000000e-01 lr 2.000000e-01 train accuracy: 0.080000 val
accuracy: 0.112000
best train accuracy achieved: 1.000000
```



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:

The five-layer network seems to be significantly more effected by initialization than the three layer one. This can be seen in our hyperparameter tuning where we get high training accuracies for multiple values of weight_scale in the three layer network (1e-2 and 1e-1) while for the five layer network only one value gives above 20% accuracy on the training set.

The reason for this could be vanishing or exploding gradients as more layers are added to the network. The more layers in the network, the more gradients are multiplied during backpropagation to get gradients for say the first layer parameters. Repeatedly mutliplying by a very small quanitity could make these gradients almost zero, halting learning. Similarly, multiplying by large quanitites repeatedly could explode the gradient and increase step sizes significanlty, causing the parameters to overshoot optimal points.

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.

[15]:
```
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
```

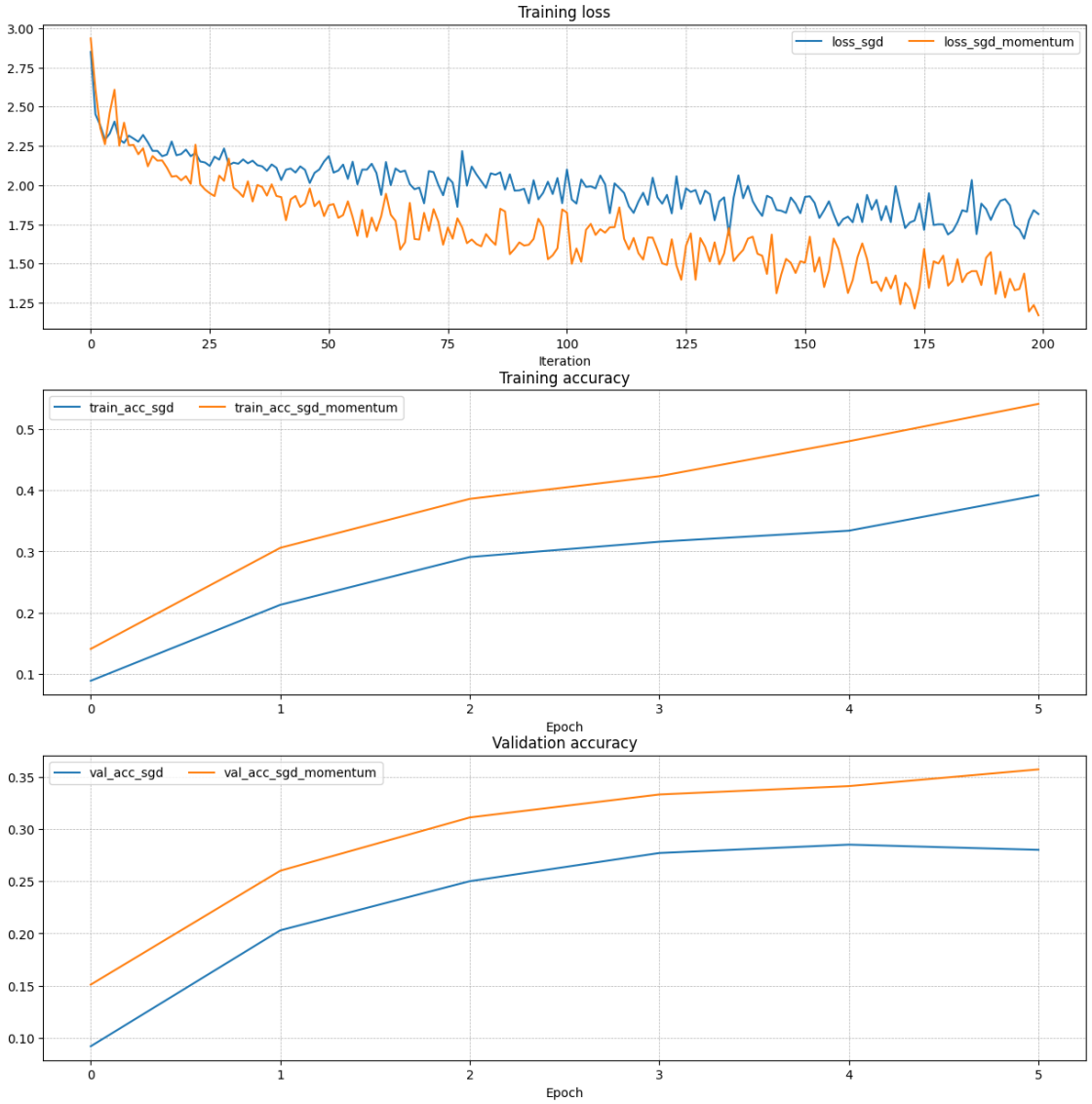
Once you have done so, run the following to train a six-layer network with both SGD and SGD+momentum. You should see the SGD+momentum update rule converge faster.

[16]:
```
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 / 200) loss: 2.848659
(Epoch 0 / 5) train acc: 0.089000; val_acc: 0.092000
(Iteration 11 / 200) loss: 2.276126
(Iteration 21 / 200) loss: 2.226377
(Iteration 31 / 200) loss: 2.143061
(Epoch 1 / 5) train acc: 0.213000; val_acc: 0.203000
(Iteration 41 / 200) loss: 2.032144
(Iteration 51 / 200) loss: 2.184910
(Iteration 61 / 200) loss: 2.077225
(Iteration 71 / 200) loss: 1.883723
(Epoch 2 / 5) train acc: 0.291000; val_acc: 0.250000
(Iteration 81 / 200) loss: 2.118058
(Iteration 91 / 200) loss: 1.966213
(Iteration 101 / 200) loss: 2.099373
(Iteration 111 / 200) loss: 2.010937
(Epoch 3 / 5) train acc: 0.316000; val_acc: 0.277000
(Iteration 121 / 200) loss: 1.881235
(Iteration 131 / 200) loss: 1.941409
(Iteration 141 / 200) loss: 1.844409
(Iteration 151 / 200) loss: 1.925058
(Epoch 4 / 5) train acc: 0.334000; val_acc: 0.285000
(Iteration 161 / 200) loss: 1.762348
(Iteration 171 / 200) loss: 1.855890
(Iteration 181 / 200) loss: 1.684910
(Iteration 191 / 200) loss: 1.847455
(Epoch 5 / 5) train acc: 0.392000; val_acc: 0.280000
Running with  sgd_momentum
```

```
(Iteration 1 / 200) loss: 2.935175
(Epoch 0 / 5) train acc: 0.141000; val_acc: 0.151000
(Iteration 11 / 200) loss: 2.196226
(Iteration 21 / 200) loss: 2.056858
(Iteration 31 / 200) loss: 1.983330
(Epoch 1 / 5) train acc: 0.306000; val_acc: 0.260000
(Iteration 41 / 200) loss: 1.923466
(Iteration 51 / 200) loss: 1.871454
(Iteration 61 / 200) loss: 1.708270
(Iteration 71 / 200) loss: 1.822766
(Epoch 2 / 5) train acc: 0.386000; val_acc: 0.311000
(Iteration 81 / 200) loss: 1.653309
(Iteration 91 / 200) loss: 1.634880
(Iteration 101 / 200) loss: 1.824510
(Iteration 111 / 200) loss: 1.731618
(Epoch 3 / 5) train acc: 0.423000; val_acc: 0.333000
(Iteration 121 / 200) loss: 1.500925
(Iteration 131 / 200) loss: 1.512699
(Iteration 141 / 200) loss: 1.562248
(Iteration 151 / 200) loss: 1.505510
(Epoch 4 / 5) train acc: 0.480000; val_acc: 0.341000
(Iteration 161 / 200) loss: 1.394740
(Iteration 171 / 200) loss: 1.239678
(Iteration 181 / 200) loss: 1.358908
(Iteration 191 / 200) loss: 1.306356
(Epoch 5 / 5) train acc: 0.541000; val_acc: 0.357000
```



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.

[18]:
```
# 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
```

[23]:
```
# 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
```

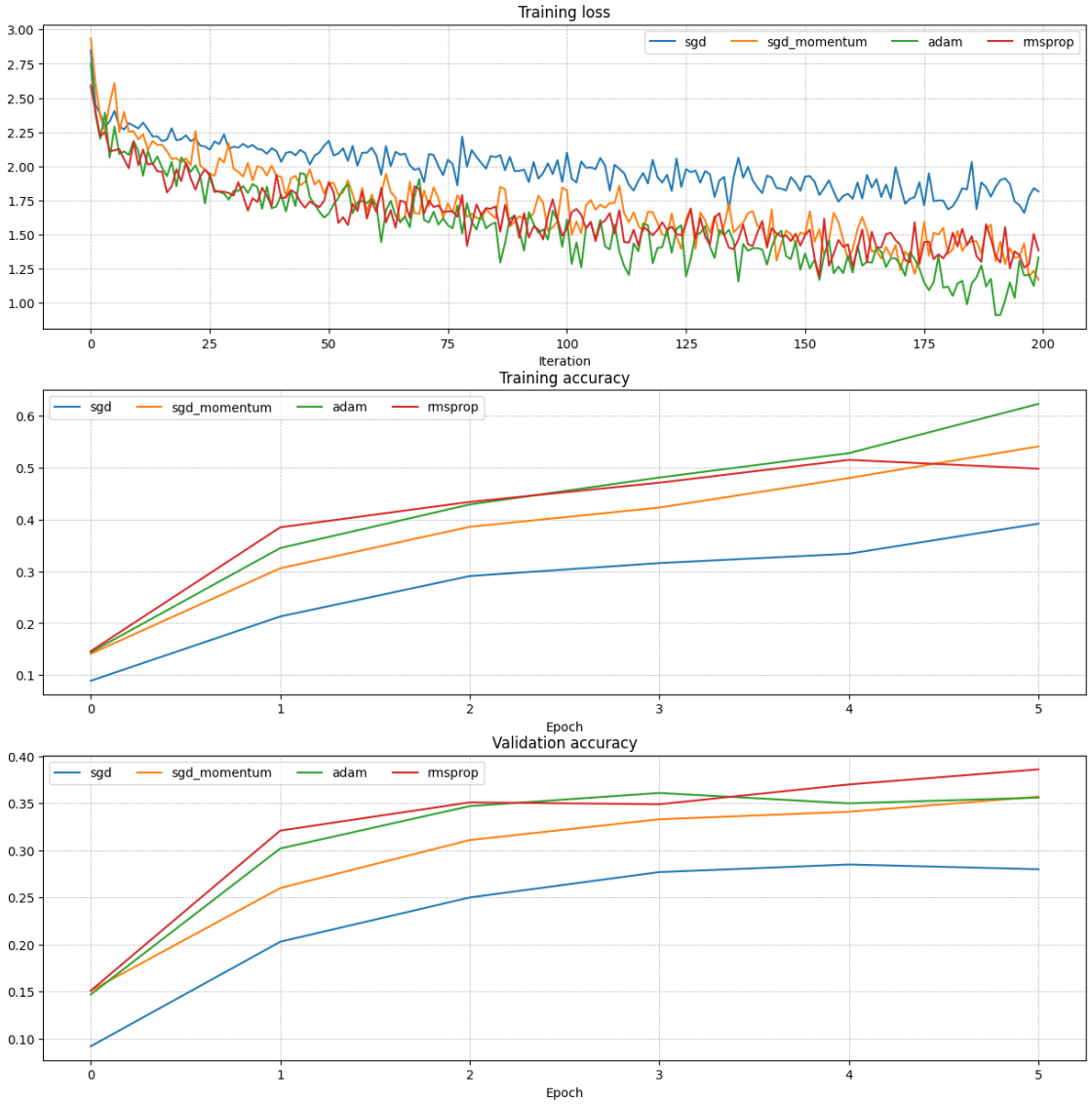
Once you have debugged your RMSProp and Adam implementations, run the following to train a pair of deep networks using these new update rules:

[24]:
```
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: 2.750324
(Epoch 0 / 5) train acc: 0.144000; val_acc: 0.147000
(Iteration 11 / 200) loss: 2.111620
(Iteration 21 / 200) loss: 2.021784
(Iteration 31 / 200) loss: 1.820050
(Epoch 1 / 5) train acc: 0.345000; val_acc: 0.302000
(Iteration 41 / 200) loss: 1.792357
(Iteration 51 / 200) loss: 1.654000
(Iteration 61 / 200) loss: 1.713413
(Iteration 71 / 200) loss: 1.607876
(Epoch 2 / 5) train acc: 0.429000; val_acc: 0.347000
(Iteration 81 / 200) loss: 1.593302
(Iteration 91 / 200) loss: 1.526982
(Iteration 101 / 200) loss: 1.610553
(Iteration 111 / 200) loss: 1.574223
(Epoch 3 / 5) train acc: 0.481000; val_acc: 0.361000
(Iteration 121 / 200) loss: 1.408578
(Iteration 131 / 200) loss: 1.412894
(Iteration 141 / 200) loss: 1.400828
(Iteration 151 / 200) loss: 1.362720
(Epoch 4 / 5) train acc: 0.528000; val_acc: 0.350000
(Iteration 161 / 200) loss: 1.222563
(Iteration 171 / 200) loss: 1.280718
(Iteration 181 / 200) loss: 1.119981
(Iteration 191 / 200) loss: 0.910158
(Epoch 5 / 5) train acc: 0.623000; val_acc: 0.356000

Running with  rmsprop
(Iteration 1 / 200) loss: 2.591508
(Epoch 0 / 5) train acc: 0.146000; val_acc: 0.151000
(Iteration 11 / 200) loss: 2.007767
```

```
(Iteration 21 / 200) loss: 2.026447
(Iteration 31 / 200) loss: 1.784996
(Epoch 1 / 5) train acc: 0.385000; val_acc: 0.321000
(Iteration 41 / 200) loss: 1.767986
(Iteration 51 / 200) loss: 1.885474
(Iteration 61 / 200) loss: 1.663920
(Iteration 71 / 200) loss: 1.654167
(Epoch 2 / 5) train acc: 0.434000; val_acc: 0.351000
(Iteration 81 / 200) loss: 1.577457
(Iteration 91 / 200) loss: 1.477054
(Iteration 101 / 200) loss: 1.542709
(Iteration 111 / 200) loss: 1.508080
(Epoch 3 / 5) train acc: 0.471000; val_acc: 0.349000
(Iteration 121 / 200) loss: 1.590659
(Iteration 131 / 200) loss: 1.425581
(Iteration 141 / 200) loss: 1.516743
(Iteration 151 / 200) loss: 1.495392
(Epoch 4 / 5) train acc: 0.515000; val_acc: 0.370000
(Iteration 161 / 200) loss: 1.265572
(Iteration 171 / 200) loss: 1.424246
(Iteration 181 / 200) loss: 1.379509
(Iteration 191 / 200) loss: 1.343727
(Epoch 5 / 5) train acc: 0.498000; val_acc: 0.386000
```



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:

As number of iterations increase, the running sum of gradients keeps increasing for AdaGrad. This being in the denominator, makes the step size very small at higher iterations.

Adam, on the other hand, maintains a convex running sum of gradients - i.e. it discounts previous gradients by a factor. This means that the value in the denominator does not shoot up with iterations and thus update size doesn't decay 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.

[19]:
```
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)*****

# weight_scale = [1e-2,1e-1]
# learning_rate = [2e-4,11e-4,2e-3]
# update_rule = ['sgd_momentum', 'adam', 'rmsprop']
# reg = [1e-1, 1e-2, 1, 1e1, 1e2]

weight_scale = [1e-1]
learning_rate = [2e-4]
update_rule = ['rmsprop']
reg = [1e-3]

results = {}
best_val = 0
best_solver = None
solvers = {}

for i in weight_scale:
  for j in learning_rate:
    for rule in update_rule:
      model = FullyConnectedNet(
          [100, 100, 100, 100, 100],
          weight_scale = i,
          dtype=np.float64,
          reg = reg[0],
      )
```

```
      solver = Solver(
          model,
          data,
          print_every=10,
          num_epochs=20,
          batch_size=25,
          update_rule= rule,
          optim_config={"learning_rate": j},
      )
      solver.train()

      solvers[(i,j,rule)] = solver

      if solver.best_val_acc > best_val:
          best_model = model
          best_solver = solver
          best_val = solver.best_val_acc

      results[(i,j,rule)] = (solver.train_acc_history[len(solver.
 ↪train_acc_history) - 1], solver.best_val_acc)

# Print out results.
for i, j, rule in sorted(results):
    train_accuracy, val_accuracy = results[(i,j,rule)]
    print('weight_scale  %e lr %e rule %s train accuracy: %f val accuracy: %f'␣
 ↪% (
                i, j, rule, train_accuracy, val_accuracy))

print('best val accuracy achieved: %f' % best_val)

# Print out results.
for i, j, rule in sorted(results):
    train_accuracy, val_accuracy = results[(i,j,rule)]
    print('weight_scale  %e lr %e rule %s train accuracy: %f val accuracy: %f'␣
 ↪% (
                i, j, rule, train_accuracy, val_accuracy))

print('best val accuracy achieved: %f' % best_val)

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')
```

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

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

plt.show()

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

```
(Iteration 1 / 39200) loss: 99.066483
(Epoch 0 / 20) train acc: 0.106000; val_acc: 0.120000
(Iteration 11 / 39200) loss: 22.709656
(Iteration 21 / 39200) loss: 20.675154
(Iteration 31 / 39200) loss: 21.862113
(Iteration 41 / 39200) loss: 17.847248
(Iteration 51 / 39200) loss: 12.737599
(Iteration 61 / 39200) loss: 15.188730
(Iteration 71 / 39200) loss: 14.560160
(Iteration 81 / 39200) loss: 14.545472
(Iteration 91 / 39200) loss: 13.461726
(Iteration 101 / 39200) loss: 8.077403
(Iteration 111 / 39200) loss: 10.187128
(Iteration 121 / 39200) loss: 11.901740
(Iteration 131 / 39200) loss: 8.659592
(Iteration 141 / 39200) loss: 8.604644
(Iteration 151 / 39200) loss: 7.843852
(Iteration 161 / 39200) loss: 9.953966
(Iteration 171 / 39200) loss: 8.347199
(Iteration 181 / 39200) loss: 6.805345
(Iteration 191 / 39200) loss: 9.136222
(Iteration 201 / 39200) loss: 8.536735
(Iteration 211 / 39200) loss: 6.522159
(Iteration 221 / 39200) loss: 8.543658
(Iteration 231 / 39200) loss: 8.153820
(Iteration 241 / 39200) loss: 8.362527
(Iteration 251 / 39200) loss: 8.274852
(Iteration 261 / 39200) loss: 7.658585
(Iteration 271 / 39200) loss: 7.240683
(Iteration 281 / 39200) loss: 6.737665
```

```
(Iteration 291 / 39200) loss: 7.834007
(Iteration 301 / 39200) loss: 5.444890
(Iteration 311 / 39200) loss: 7.612860
(Iteration 321 / 39200) loss: 5.975837
(Iteration 331 / 39200) loss: 8.177976
(Iteration 341 / 39200) loss: 5.446091
(Iteration 351 / 39200) loss: 5.937114
(Iteration 361 / 39200) loss: 7.391316
(Iteration 371 / 39200) loss: 5.614235
(Iteration 381 / 39200) loss: 4.134787
(Iteration 391 / 39200) loss: 6.092285
(Iteration 401 / 39200) loss: 6.137340
(Iteration 411 / 39200) loss: 4.671179
(Iteration 421 / 39200) loss: 5.368636
(Iteration 431 / 39200) loss: 5.751093
(Iteration 441 / 39200) loss: 5.231289
(Iteration 451 / 39200) loss: 5.044494
(Iteration 461 / 39200) loss: 5.433750
(Iteration 471 / 39200) loss: 6.343196
(Iteration 481 / 39200) loss: 5.727665
(Iteration 491 / 39200) loss: 5.791197
(Iteration 501 / 39200) loss: 5.493343
(Iteration 511 / 39200) loss: 4.644368
(Iteration 521 / 39200) loss: 4.500774
(Iteration 531 / 39200) loss: 5.595597
(Iteration 541 / 39200) loss: 4.352607
(Iteration 551 / 39200) loss: 4.926693
(Iteration 561 / 39200) loss: 5.195593
(Iteration 571 / 39200) loss: 5.404572
(Iteration 581 / 39200) loss: 4.028517
(Iteration 591 / 39200) loss: 4.793100
(Iteration 601 / 39200) loss: 5.428331
(Iteration 611 / 39200) loss: 4.338152
(Iteration 621 / 39200) loss: 4.769717
(Iteration 631 / 39200) loss: 4.910484
(Iteration 641 / 39200) loss: 4.840745
(Iteration 651 / 39200) loss: 3.854396
(Iteration 661 / 39200) loss: 4.093547
(Iteration 671 / 39200) loss: 4.632972
(Iteration 681 / 39200) loss: 4.960813
(Iteration 691 / 39200) loss: 4.684751
(Iteration 701 / 39200) loss: 4.611954
(Iteration 711 / 39200) loss: 4.214837
(Iteration 721 / 39200) loss: 4.859827
(Iteration 731 / 39200) loss: 4.396279
(Iteration 741 / 39200) loss: 4.704771
(Iteration 751 / 39200) loss: 4.323321
(Iteration 761 / 39200) loss: 4.010198
```

```
(Iteration 771 / 39200) loss: 4.000384
(Iteration 781 / 39200) loss: 4.099667
(Iteration 791 / 39200) loss: 3.897195
(Iteration 801 / 39200) loss: 4.086688
(Iteration 811 / 39200) loss: 4.927335
(Iteration 821 / 39200) loss: 3.905058
(Iteration 831 / 39200) loss: 4.329212
(Iteration 841 / 39200) loss: 3.850269
(Iteration 851 / 39200) loss: 4.340304
(Iteration 861 / 39200) loss: 4.112562
(Iteration 871 / 39200) loss: 4.233421
(Iteration 881 / 39200) loss: 4.509777
(Iteration 891 / 39200) loss: 4.162288
(Iteration 901 / 39200) loss: 4.382384
(Iteration 911 / 39200) loss: 4.355432
(Iteration 921 / 39200) loss: 4.013184
(Iteration 931 / 39200) loss: 3.584612
(Iteration 941 / 39200) loss: 4.185483
(Iteration 951 / 39200) loss: 3.914465
(Iteration 961 / 39200) loss: 3.721438
(Iteration 971 / 39200) loss: 3.658791
(Iteration 981 / 39200) loss: 4.053823
(Iteration 991 / 39200) loss: 3.665663
(Iteration 1001 / 39200) loss: 4.168881
(Iteration 1011 / 39200) loss: 3.325995
(Iteration 1021 / 39200) loss: 4.203191
(Iteration 1031 / 39200) loss: 3.875087
(Iteration 1041 / 39200) loss: 3.777177
(Iteration 1051 / 39200) loss: 3.468635
(Iteration 1061 / 39200) loss: 4.388091
(Iteration 1071 / 39200) loss: 3.904867
(Iteration 1081 / 39200) loss: 3.672351
(Iteration 1091 / 39200) loss: 3.711569
(Iteration 1101 / 39200) loss: 3.860511
(Iteration 1111 / 39200) loss: 3.843943
(Iteration 1121 / 39200) loss: 4.286394
(Iteration 1131 / 39200) loss: 3.937817
(Iteration 1141 / 39200) loss: 3.706486
(Iteration 1151 / 39200) loss: 3.807866
(Iteration 1161 / 39200) loss: 3.336873
(Iteration 1171 / 39200) loss: 3.976049
(Iteration 1181 / 39200) loss: 3.604794
(Iteration 1191 / 39200) loss: 3.536875
(Iteration 1201 / 39200) loss: 3.919321
(Iteration 1211 / 39200) loss: 3.789133
(Iteration 1221 / 39200) loss: 3.751566
(Iteration 1231 / 39200) loss: 3.943184
(Iteration 1241 / 39200) loss: 3.805813
```

```
(Iteration 1251 / 39200) loss: 3.793661
(Iteration 1261 / 39200) loss: 3.864892
(Iteration 1271 / 39200) loss: 4.265041
(Iteration 1281 / 39200) loss: 4.081141
(Iteration 1291 / 39200) loss: 4.105828
(Iteration 1301 / 39200) loss: 3.925557
(Iteration 1311 / 39200) loss: 3.867812
(Iteration 1321 / 39200) loss: 3.773316
(Iteration 1331 / 39200) loss: 3.829335
(Iteration 1341 / 39200) loss: 3.915135
(Iteration 1351 / 39200) loss: 3.306984
(Iteration 1361 / 39200) loss: 3.807694
(Iteration 1371 / 39200) loss: 3.467240
(Iteration 1381 / 39200) loss: 4.035638
(Iteration 1391 / 39200) loss: 3.719980
(Iteration 1401 / 39200) loss: 3.560407
(Iteration 1411 / 39200) loss: 3.452827
(Iteration 1421 / 39200) loss: 3.798569
(Iteration 1431 / 39200) loss: 3.621083
(Iteration 1441 / 39200) loss: 3.415748
(Iteration 1451 / 39200) loss: 3.553779
(Iteration 1461 / 39200) loss: 3.967624
(Iteration 1471 / 39200) loss: 3.670744
(Iteration 1481 / 39200) loss: 3.825665
(Iteration 1491 / 39200) loss: 3.663224
(Iteration 1501 / 39200) loss: 3.363054
(Iteration 1511 / 39200) loss: 3.506741
(Iteration 1521 / 39200) loss: 3.721022
(Iteration 1531 / 39200) loss: 3.483459
(Iteration 1541 / 39200) loss: 3.677590
(Iteration 1551 / 39200) loss: 3.956596
(Iteration 1561 / 39200) loss: 3.347833
(Iteration 1571 / 39200) loss: 3.645669
(Iteration 1581 / 39200) loss: 3.590703
(Iteration 1591 / 39200) loss: 3.183090
(Iteration 1601 / 39200) loss: 3.778267
(Iteration 1611 / 39200) loss: 3.369473
(Iteration 1621 / 39200) loss: 3.645844
(Iteration 1631 / 39200) loss: 3.678416
(Iteration 1641 / 39200) loss: 3.631044
(Iteration 1651 / 39200) loss: 4.508775
(Iteration 1661 / 39200) loss: 3.329790
(Iteration 1671 / 39200) loss: 3.745312
(Iteration 1681 / 39200) loss: 3.547727
(Iteration 1691 / 39200) loss: 3.622025
(Iteration 1701 / 39200) loss: 3.588293
(Iteration 1711 / 39200) loss: 3.969219
(Iteration 1721 / 39200) loss: 3.498934
```

```
(Iteration 1731 / 39200) loss: 3.739279
(Iteration 1741 / 39200) loss: 3.278652
(Iteration 1751 / 39200) loss: 3.834425
(Iteration 1761 / 39200) loss: 3.907358
(Iteration 1771 / 39200) loss: 3.676499
(Iteration 1781 / 39200) loss: 3.875045
(Iteration 1791 / 39200) loss: 3.636906
(Iteration 1801 / 39200) loss: 3.359120
(Iteration 1811 / 39200) loss: 3.339158
(Iteration 1821 / 39200) loss: 3.513733
(Iteration 1831 / 39200) loss: 3.987263
(Iteration 1841 / 39200) loss: 3.559225
(Iteration 1851 / 39200) loss: 3.403759
(Iteration 1861 / 39200) loss: 3.901932
(Iteration 1871 / 39200) loss: 3.564846
(Iteration 1881 / 39200) loss: 3.829647
(Iteration 1891 / 39200) loss: 3.330778
(Iteration 1901 / 39200) loss: 3.401759
(Iteration 1911 / 39200) loss: 3.432775
(Iteration 1921 / 39200) loss: 3.474836
(Iteration 1931 / 39200) loss: 3.597854
(Iteration 1941 / 39200) loss: 3.872285
(Iteration 1951 / 39200) loss: 3.856918
(Epoch 1 / 20) train acc: 0.304000; val_acc: 0.286000
(Iteration 1961 / 39200) loss: 3.958023
(Iteration 1971 / 39200) loss: 3.676887
(Iteration 1981 / 39200) loss: 3.796007
(Iteration 1991 / 39200) loss: 3.650885
(Iteration 2001 / 39200) loss: 3.572168
(Iteration 2011 / 39200) loss: 3.472626
(Iteration 2021 / 39200) loss: 3.491947
(Iteration 2031 / 39200) loss: 3.391439
(Iteration 2041 / 39200) loss: 3.768969
(Iteration 2051 / 39200) loss: 4.118340
(Iteration 2061 / 39200) loss: 3.519752
(Iteration 2071 / 39200) loss: 3.461992
(Iteration 2081 / 39200) loss: 3.394509
(Iteration 2091 / 39200) loss: 2.957800
(Iteration 2101 / 39200) loss: 3.626104
(Iteration 2111 / 39200) loss: 3.561818
(Iteration 2121 / 39200) loss: 3.494445
(Iteration 2131 / 39200) loss: 3.239896
(Iteration 2141 / 39200) loss: 3.228767
(Iteration 2151 / 39200) loss: 3.550488
(Iteration 2161 / 39200) loss: 3.372914
(Iteration 2171 / 39200) loss: 3.705276
(Iteration 2181 / 39200) loss: 3.685687
(Iteration 2191 / 39200) loss: 3.336023
```

```
(Iteration 2201 / 39200) loss: 3.202950
(Iteration 2211 / 39200) loss: 3.404291
(Iteration 2221 / 39200) loss: 3.635985
(Iteration 2231 / 39200) loss: 3.390627
(Iteration 2241 / 39200) loss: 3.464734
(Iteration 2251 / 39200) loss: 3.357949
(Iteration 2261 / 39200) loss: 3.643899
(Iteration 2271 / 39200) loss: 3.517263
(Iteration 2281 / 39200) loss: 3.415872
(Iteration 2291 / 39200) loss: 3.404417
(Iteration 2301 / 39200) loss: 3.173323
(Iteration 2311 / 39200) loss: 3.261543
(Iteration 2321 / 39200) loss: 3.587515
(Iteration 2331 / 39200) loss: 3.650931
(Iteration 2341 / 39200) loss: 3.715480
(Iteration 2351 / 39200) loss: 3.346249
(Iteration 2361 / 39200) loss: 3.690566
(Iteration 2371 / 39200) loss: 3.231804
(Iteration 2381 / 39200) loss: 3.425917
(Iteration 2391 / 39200) loss: 3.680449
(Iteration 2401 / 39200) loss: 3.316268
(Iteration 2411 / 39200) loss: 3.209212
(Iteration 2421 / 39200) loss: 3.468700
(Iteration 2431 / 39200) loss: 3.338334
(Iteration 2441 / 39200) loss: 3.325479
(Iteration 2451 / 39200) loss: 3.603242
(Iteration 2461 / 39200) loss: 3.406795
(Iteration 2471 / 39200) loss: 3.554299
(Iteration 2481 / 39200) loss: 3.495827
(Iteration 2491 / 39200) loss: 3.259254
(Iteration 2501 / 39200) loss: 3.308806
(Iteration 2511 / 39200) loss: 2.831432
(Iteration 2521 / 39200) loss: 3.628340
(Iteration 2531 / 39200) loss: 2.992466
(Iteration 2541 / 39200) loss: 3.201069
(Iteration 2551 / 39200) loss: 3.248490
(Iteration 2561 / 39200) loss: 3.316525
(Iteration 2571 / 39200) loss: 3.726629
(Iteration 2581 / 39200) loss: 3.283676
(Iteration 2591 / 39200) loss: 3.077828
(Iteration 2601 / 39200) loss: 3.737090
(Iteration 2611 / 39200) loss: 3.530611
(Iteration 2621 / 39200) loss: 3.057874
(Iteration 2631 / 39200) loss: 3.714560
(Iteration 2641 / 39200) loss: 3.201756
(Iteration 2651 / 39200) loss: 3.484801
(Iteration 2661 / 39200) loss: 3.202441
(Iteration 2671 / 39200) loss: 3.597742
```

```
(Iteration 2681 / 39200) loss: 3.617522
(Iteration 2691 / 39200) loss: 3.563421
(Iteration 2701 / 39200) loss: 3.146731
(Iteration 2711 / 39200) loss: 3.520235
(Iteration 2721 / 39200) loss: 3.500937
(Iteration 2731 / 39200) loss: 3.254637
(Iteration 2741 / 39200) loss: 3.424551
(Iteration 2751 / 39200) loss: 3.293291
(Iteration 2761 / 39200) loss: 3.317406
(Iteration 2771 / 39200) loss: 3.017579
(Iteration 2781 / 39200) loss: 3.336970
(Iteration 2791 / 39200) loss: 3.370415
(Iteration 2801 / 39200) loss: 3.548015
(Iteration 2811 / 39200) loss: 3.113301
(Iteration 2821 / 39200) loss: 3.318172
(Iteration 2831 / 39200) loss: 3.780873
(Iteration 2841 / 39200) loss: 3.374307
(Iteration 2851 / 39200) loss: 3.249035
(Iteration 2861 / 39200) loss: 3.132593
(Iteration 2871 / 39200) loss: 3.757229
(Iteration 2881 / 39200) loss: 3.527405
(Iteration 2891 / 39200) loss: 3.164695
(Iteration 2901 / 39200) loss: 3.181451
(Iteration 2911 / 39200) loss: 3.300594
(Iteration 2921 / 39200) loss: 3.478849
(Iteration 2931 / 39200) loss: 3.647190
(Iteration 2941 / 39200) loss: 3.229988
(Iteration 2951 / 39200) loss: 2.991609
(Iteration 2961 / 39200) loss: 3.363675
(Iteration 2971 / 39200) loss: 3.643669
(Iteration 2981 / 39200) loss: 3.245948
(Iteration 2991 / 39200) loss: 3.567310
(Iteration 3001 / 39200) loss: 3.257107
(Iteration 3011 / 39200) loss: 3.180605
(Iteration 3021 / 39200) loss: 3.310623
(Iteration 3031 / 39200) loss: 3.628399
(Iteration 3041 / 39200) loss: 3.313262
(Iteration 3051 / 39200) loss: 3.343241
(Iteration 3061 / 39200) loss: 3.367183
(Iteration 3071 / 39200) loss: 3.606935
(Iteration 3081 / 39200) loss: 3.113232
(Iteration 3091 / 39200) loss: 3.460724
(Iteration 3101 / 39200) loss: 3.250522
(Iteration 3111 / 39200) loss: 3.637511
(Iteration 3121 / 39200) loss: 3.600554
(Iteration 3131 / 39200) loss: 3.332266
(Iteration 3141 / 39200) loss: 3.654793
(Iteration 3151 / 39200) loss: 3.060125
```

```
(Iteration 3161 / 39200) loss: 3.506208
(Iteration 3171 / 39200) loss: 3.326693
(Iteration 3181 / 39200) loss: 3.315939
(Iteration 3191 / 39200) loss: 3.505987
(Iteration 3201 / 39200) loss: 2.896080
(Iteration 3211 / 39200) loss: 2.944398
(Iteration 3221 / 39200) loss: 3.267935
(Iteration 3231 / 39200) loss: 3.365072
(Iteration 3241 / 39200) loss: 2.755435
(Iteration 3251 / 39200) loss: 2.862291
(Iteration 3261 / 39200) loss: 3.169434
(Iteration 3271 / 39200) loss: 3.263836
(Iteration 3281 / 39200) loss: 3.090692
(Iteration 3291 / 39200) loss: 3.208387
(Iteration 3301 / 39200) loss: 3.076489
(Iteration 3311 / 39200) loss: 3.634881
(Iteration 3321 / 39200) loss: 3.633265
(Iteration 3331 / 39200) loss: 3.463467
(Iteration 3341 / 39200) loss: 3.219332
(Iteration 3351 / 39200) loss: 3.271471
(Iteration 3361 / 39200) loss: 3.144222
(Iteration 3371 / 39200) loss: 3.173729
(Iteration 3381 / 39200) loss: 3.098815
(Iteration 3391 / 39200) loss: 3.262931
(Iteration 3401 / 39200) loss: 2.994439
(Iteration 3411 / 39200) loss: 3.169739
(Iteration 3421 / 39200) loss: 3.041474
(Iteration 3431 / 39200) loss: 3.011222
(Iteration 3441 / 39200) loss: 3.331652
(Iteration 3451 / 39200) loss: 3.374555
(Iteration 3461 / 39200) loss: 3.973137
(Iteration 3471 / 39200) loss: 3.204457
(Iteration 3481 / 39200) loss: 3.221455
(Iteration 3491 / 39200) loss: 3.260332
(Iteration 3501 / 39200) loss: 3.151810
(Iteration 3511 / 39200) loss: 2.953399
(Iteration 3521 / 39200) loss: 3.306452
(Iteration 3531 / 39200) loss: 3.236673
(Iteration 3541 / 39200) loss: 3.188823
(Iteration 3551 / 39200) loss: 3.177617
(Iteration 3561 / 39200) loss: 3.197915
(Iteration 3571 / 39200) loss: 2.912388
(Iteration 3581 / 39200) loss: 3.100141
(Iteration 3591 / 39200) loss: 3.068696
(Iteration 3601 / 39200) loss: 3.037905
(Iteration 3611 / 39200) loss: 3.282986
(Iteration 3621 / 39200) loss: 3.213858
(Iteration 3631 / 39200) loss: 2.745950
```

```
(Iteration 3641 / 39200) loss: 3.195999
(Iteration 3651 / 39200) loss: 2.977024
(Iteration 3661 / 39200) loss: 3.163576
(Iteration 3671 / 39200) loss: 2.959863
(Iteration 3681 / 39200) loss: 3.186861
(Iteration 3691 / 39200) loss: 3.534795
(Iteration 3701 / 39200) loss: 3.061955
(Iteration 3711 / 39200) loss: 3.109376
(Iteration 3721 / 39200) loss: 2.919569
(Iteration 3731 / 39200) loss: 3.360034
(Iteration 3741 / 39200) loss: 3.144287
(Iteration 3751 / 39200) loss: 3.034439
(Iteration 3761 / 39200) loss: 3.226534
(Iteration 3771 / 39200) loss: 3.021007
(Iteration 3781 / 39200) loss: 3.313318
(Iteration 3791 / 39200) loss: 3.219617
(Iteration 3801 / 39200) loss: 3.181126
(Iteration 3811 / 39200) loss: 3.624106
(Iteration 3821 / 39200) loss: 2.905401
(Iteration 3831 / 39200) loss: 2.978469
(Iteration 3841 / 39200) loss: 3.859760
(Iteration 3851 / 39200) loss: 2.948892
(Iteration 3861 / 39200) loss: 3.228375
(Iteration 3871 / 39200) loss: 3.356159
(Iteration 3881 / 39200) loss: 2.967113
(Iteration 3891 / 39200) loss: 3.079693
(Iteration 3901 / 39200) loss: 3.163086
(Iteration 3911 / 39200) loss: 3.177803
(Epoch 2 / 20) train acc: 0.413000; val_acc: 0.360000
(Iteration 3921 / 39200) loss: 3.143724
(Iteration 3931 / 39200) loss: 3.349830
(Iteration 3941 / 39200) loss: 3.014734
(Iteration 3951 / 39200) loss: 2.990883
(Iteration 3961 / 39200) loss: 2.918077
(Iteration 3971 / 39200) loss: 3.322609
(Iteration 3981 / 39200) loss: 2.900891
(Iteration 3991 / 39200) loss: 3.387030
(Iteration 4001 / 39200) loss: 3.234016
(Iteration 4011 / 39200) loss: 3.103044
(Iteration 4021 / 39200) loss: 3.253325
(Iteration 4031 / 39200) loss: 3.113561
(Iteration 4041 / 39200) loss: 3.388844
(Iteration 4051 / 39200) loss: 3.364939
(Iteration 4061 / 39200) loss: 3.394550
(Iteration 4071 / 39200) loss: 3.087030
(Iteration 4081 / 39200) loss: 3.011359
(Iteration 4091 / 39200) loss: 3.026610
(Iteration 4101 / 39200) loss: 3.300210
```

```
(Iteration 4111 / 39200) loss: 3.076395
(Iteration 4121 / 39200) loss: 2.764135
(Iteration 4131 / 39200) loss: 2.957695
(Iteration 4141 / 39200) loss: 2.997691
(Iteration 4151 / 39200) loss: 2.775049
(Iteration 4161 / 39200) loss: 3.203996
(Iteration 4171 / 39200) loss: 2.659900
(Iteration 4181 / 39200) loss: 3.357959
(Iteration 4191 / 39200) loss: 3.147158
(Iteration 4201 / 39200) loss: 3.463192
(Iteration 4211 / 39200) loss: 2.834022
(Iteration 4221 / 39200) loss: 3.085433
(Iteration 4231 / 39200) loss: 3.192867
(Iteration 4241 / 39200) loss: 3.025811
(Iteration 4251 / 39200) loss: 2.849766
(Iteration 4261 / 39200) loss: 2.925416
(Iteration 4271 / 39200) loss: 3.070113
(Iteration 4281 / 39200) loss: 3.356141
(Iteration 4291 / 39200) loss: 2.957518
(Iteration 4301 / 39200) loss: 3.144765
(Iteration 4311 / 39200) loss: 2.863002
(Iteration 4321 / 39200) loss: 2.963886
(Iteration 4331 / 39200) loss: 2.839842
(Iteration 4341 / 39200) loss: 3.137405
(Iteration 4351 / 39200) loss: 2.961407
(Iteration 4361 / 39200) loss: 3.111503
(Iteration 4371 / 39200) loss: 3.164762
(Iteration 4381 / 39200) loss: 3.096641
(Iteration 4391 / 39200) loss: 3.067474
(Iteration 4401 / 39200) loss: 3.316385
(Iteration 4411 / 39200) loss: 2.899217
(Iteration 4421 / 39200) loss: 2.807928
(Iteration 4431 / 39200) loss: 2.716715
(Iteration 4441 / 39200) loss: 2.768606
(Iteration 4451 / 39200) loss: 3.049243
(Iteration 4461 / 39200) loss: 3.004041
(Iteration 4471 / 39200) loss: 2.780600
(Iteration 4481 / 39200) loss: 3.124626
(Iteration 4491 / 39200) loss: 3.163627
(Iteration 4501 / 39200) loss: 2.932212
(Iteration 4511 / 39200) loss: 3.134955
(Iteration 4521 / 39200) loss: 2.991467
(Iteration 4531 / 39200) loss: 3.645719
(Iteration 4541 / 39200) loss: 2.838959
(Iteration 4551 / 39200) loss: 3.118660
(Iteration 4561 / 39200) loss: 3.133131
(Iteration 4571 / 39200) loss: 3.318199
(Iteration 4581 / 39200) loss: 3.194116
```

```
(Iteration 4591 / 39200) loss: 3.100148
(Iteration 4601 / 39200) loss: 3.199922
(Iteration 4611 / 39200) loss: 3.142079
(Iteration 4621 / 39200) loss: 2.699555
(Iteration 4631 / 39200) loss: 3.066019
(Iteration 4641 / 39200) loss: 2.984356
(Iteration 4651 / 39200) loss: 3.426911
(Iteration 4661 / 39200) loss: 2.895992
(Iteration 4671 / 39200) loss: 3.199139
(Iteration 4681 / 39200) loss: 2.998035
(Iteration 4691 / 39200) loss: 3.030889
(Iteration 4701 / 39200) loss: 2.763286
(Iteration 4711 / 39200) loss: 3.049433
(Iteration 4721 / 39200) loss: 2.996206
(Iteration 4731 / 39200) loss: 3.010482
(Iteration 4741 / 39200) loss: 3.002391
(Iteration 4751 / 39200) loss: 2.679563
(Iteration 4761 / 39200) loss: 3.172705
(Iteration 4771 / 39200) loss: 2.804650
(Iteration 4781 / 39200) loss: 3.246484
(Iteration 4791 / 39200) loss: 2.921445
(Iteration 4801 / 39200) loss: 2.687628
(Iteration 4811 / 39200) loss: 2.961328
(Iteration 4821 / 39200) loss: 2.748666
(Iteration 4831 / 39200) loss: 2.889919
(Iteration 4841 / 39200) loss: 2.881746
(Iteration 4851 / 39200) loss: 3.167180
(Iteration 4861 / 39200) loss: 2.811347
(Iteration 4871 / 39200) loss: 2.983625
(Iteration 4881 / 39200) loss: 3.468570
(Iteration 4891 / 39200) loss: 2.859095
(Iteration 4901 / 39200) loss: 2.517816
(Iteration 4911 / 39200) loss: 2.963694
(Iteration 4921 / 39200) loss: 2.743596
(Iteration 4931 / 39200) loss: 2.908018
(Iteration 4941 / 39200) loss: 3.227145
(Iteration 4951 / 39200) loss: 3.149797
(Iteration 4961 / 39200) loss: 3.244110
(Iteration 4971 / 39200) loss: 2.865397
(Iteration 4981 / 39200) loss: 2.958631
(Iteration 4991 / 39200) loss: 2.604656
(Iteration 5001 / 39200) loss: 3.315662
(Iteration 5011 / 39200) loss: 3.085550
(Iteration 5021 / 39200) loss: 2.680216
(Iteration 5031 / 39200) loss: 3.112466
(Iteration 5041 / 39200) loss: 2.878205
(Iteration 5051 / 39200) loss: 3.270965
(Iteration 5061 / 39200) loss: 3.209999
```

```
(Iteration 5071 / 39200) loss: 2.976719
(Iteration 5081 / 39200) loss: 2.634405
(Iteration 5091 / 39200) loss: 2.697229
(Iteration 5101 / 39200) loss: 3.103943
(Iteration 5111 / 39200) loss: 2.956033
(Iteration 5121 / 39200) loss: 3.056515
(Iteration 5131 / 39200) loss: 2.838112
(Iteration 5141 / 39200) loss: 2.945908
(Iteration 5151 / 39200) loss: 3.178760
(Iteration 5161 / 39200) loss: 3.376924
(Iteration 5171 / 39200) loss: 2.886536
(Iteration 5181 / 39200) loss: 2.707459
(Iteration 5191 / 39200) loss: 2.816884
(Iteration 5201 / 39200) loss: 3.247893
(Iteration 5211 / 39200) loss: 2.656814
(Iteration 5221 / 39200) loss: 2.735064
(Iteration 5231 / 39200) loss: 2.770133
(Iteration 5241 / 39200) loss: 2.925576
(Iteration 5251 / 39200) loss: 3.128640
(Iteration 5261 / 39200) loss: 2.674942
(Iteration 5271 / 39200) loss: 3.163440
(Iteration 5281 / 39200) loss: 2.798517
(Iteration 5291 / 39200) loss: 2.924704
(Iteration 5301 / 39200) loss: 3.330764
(Iteration 5311 / 39200) loss: 2.774327
(Iteration 5321 / 39200) loss: 2.832883
(Iteration 5331 / 39200) loss: 2.512943
(Iteration 5341 / 39200) loss: 2.748319
(Iteration 5351 / 39200) loss: 2.424536
(Iteration 5361 / 39200) loss: 2.924382
(Iteration 5371 / 39200) loss: 2.658122
(Iteration 5381 / 39200) loss: 3.150389
(Iteration 5391 / 39200) loss: 2.990318
(Iteration 5401 / 39200) loss: 3.054858
(Iteration 5411 / 39200) loss: 2.534328
(Iteration 5421 / 39200) loss: 2.762755
(Iteration 5431 / 39200) loss: 2.739273
(Iteration 5441 / 39200) loss: 2.514817
(Iteration 5451 / 39200) loss: 2.856951
(Iteration 5461 / 39200) loss: 3.395466
(Iteration 5471 / 39200) loss: 3.233496
(Iteration 5481 / 39200) loss: 2.690517
(Iteration 5491 / 39200) loss: 2.983259
(Iteration 5501 / 39200) loss: 2.959093
(Iteration 5511 / 39200) loss: 3.060431
(Iteration 5521 / 39200) loss: 2.858435
(Iteration 5531 / 39200) loss: 2.579724
(Iteration 5541 / 39200) loss: 3.010489
```

```
(Iteration 5551 / 39200) loss: 2.685557
(Iteration 5561 / 39200) loss: 2.977006
(Iteration 5571 / 39200) loss: 2.728061
(Iteration 5581 / 39200) loss: 3.076733
(Iteration 5591 / 39200) loss: 3.245810
(Iteration 5601 / 39200) loss: 3.048539
(Iteration 5611 / 39200) loss: 2.881612
(Iteration 5621 / 39200) loss: 2.908685
(Iteration 5631 / 39200) loss: 2.501672
(Iteration 5641 / 39200) loss: 2.673696
(Iteration 5651 / 39200) loss: 2.986840
(Iteration 5661 / 39200) loss: 2.718804
(Iteration 5671 / 39200) loss: 2.757303
(Iteration 5681 / 39200) loss: 2.906454
(Iteration 5691 / 39200) loss: 2.574103
(Iteration 5701 / 39200) loss: 2.869740
(Iteration 5711 / 39200) loss: 2.333304
(Iteration 5721 / 39200) loss: 2.685891
(Iteration 5731 / 39200) loss: 2.798009
(Iteration 5741 / 39200) loss: 2.782509
(Iteration 5751 / 39200) loss: 2.808379
(Iteration 5761 / 39200) loss: 2.834164
(Iteration 5771 / 39200) loss: 2.825621
(Iteration 5781 / 39200) loss: 2.779649
(Iteration 5791 / 39200) loss: 3.199077
(Iteration 5801 / 39200) loss: 2.548206
(Iteration 5811 / 39200) loss: 2.638443
(Iteration 5821 / 39200) loss: 2.462294
(Iteration 5831 / 39200) loss: 2.842255
(Iteration 5841 / 39200) loss: 2.615988
(Iteration 5851 / 39200) loss: 2.635496
(Iteration 5861 / 39200) loss: 2.718714
(Iteration 5871 / 39200) loss: 2.970906
(Epoch 3 / 20) train acc: 0.453000; val_acc: 0.398000
(Iteration 5881 / 39200) loss: 2.581252
(Iteration 5891 / 39200) loss: 2.855534
(Iteration 5901 / 39200) loss: 2.363933
(Iteration 5911 / 39200) loss: 2.790253
(Iteration 5921 / 39200) loss: 2.798325
(Iteration 5931 / 39200) loss: 2.646455
(Iteration 5941 / 39200) loss: 2.879044
(Iteration 5951 / 39200) loss: 2.756371
(Iteration 5961 / 39200) loss: 2.837909
(Iteration 5971 / 39200) loss: 2.576509
(Iteration 5981 / 39200) loss: 2.666855
(Iteration 5991 / 39200) loss: 2.914472
(Iteration 6001 / 39200) loss: 2.547490
(Iteration 6011 / 39200) loss: 2.683509
```

```
(Iteration 6021 / 39200) loss: 2.665332
(Iteration 6031 / 39200) loss: 2.815693
(Iteration 6041 / 39200) loss: 2.750833
(Iteration 6051 / 39200) loss: 3.142197
(Iteration 6061 / 39200) loss: 2.794967
(Iteration 6071 / 39200) loss: 2.938106
(Iteration 6081 / 39200) loss: 2.952968
(Iteration 6091 / 39200) loss: 2.976081
(Iteration 6101 / 39200) loss: 2.572417
(Iteration 6111 / 39200) loss: 2.575109
(Iteration 6121 / 39200) loss: 2.509743
(Iteration 6131 / 39200) loss: 3.214954
(Iteration 6141 / 39200) loss: 2.762123
(Iteration 6151 / 39200) loss: 3.128581
(Iteration 6161 / 39200) loss: 2.798527
(Iteration 6171 / 39200) loss: 2.966030
(Iteration 6181 / 39200) loss: 2.953799
(Iteration 6191 / 39200) loss: 2.803569
(Iteration 6201 / 39200) loss: 2.737316
(Iteration 6211 / 39200) loss: 2.924225
(Iteration 6221 / 39200) loss: 2.650796
(Iteration 6231 / 39200) loss: 2.800247
(Iteration 6241 / 39200) loss: 2.986951
(Iteration 6251 / 39200) loss: 2.982591
(Iteration 6261 / 39200) loss: 2.753289
(Iteration 6271 / 39200) loss: 2.695369
(Iteration 6281 / 39200) loss: 2.600003
(Iteration 6291 / 39200) loss: 2.583788
(Iteration 6301 / 39200) loss: 3.274050
(Iteration 6311 / 39200) loss: 2.559434
(Iteration 6321 / 39200) loss: 2.858207
(Iteration 6331 / 39200) loss: 3.044494
(Iteration 6341 / 39200) loss: 2.886952
(Iteration 6351 / 39200) loss: 2.707070
(Iteration 6361 / 39200) loss: 2.893767
(Iteration 6371 / 39200) loss: 2.355451
(Iteration 6381 / 39200) loss: 3.183712
(Iteration 6391 / 39200) loss: 3.208100
(Iteration 6401 / 39200) loss: 2.650803
(Iteration 6411 / 39200) loss: 2.711277
(Iteration 6421 / 39200) loss: 2.847640
(Iteration 6431 / 39200) loss: 2.846311
(Iteration 6441 / 39200) loss: 2.529992
(Iteration 6451 / 39200) loss: 2.968442
(Iteration 6461 / 39200) loss: 2.758816
(Iteration 6471 / 39200) loss: 2.713959
(Iteration 6481 / 39200) loss: 2.835228
(Iteration 6491 / 39200) loss: 2.800081
```

```
(Iteration 6501 / 39200) loss: 2.845463
(Iteration 6511 / 39200) loss: 2.711255
(Iteration 6521 / 39200) loss: 2.954404
(Iteration 6531 / 39200) loss: 2.753778
(Iteration 6541 / 39200) loss: 2.569225
(Iteration 6551 / 39200) loss: 2.688621
(Iteration 6561 / 39200) loss: 3.019148
(Iteration 6571 / 39200) loss: 2.689671
(Iteration 6581 / 39200) loss: 2.597685
(Iteration 6591 / 39200) loss: 2.677446
(Iteration 6601 / 39200) loss: 3.136639
(Iteration 6611 / 39200) loss: 2.781491
(Iteration 6621 / 39200) loss: 2.707481
(Iteration 6631 / 39200) loss: 2.626001
(Iteration 6641 / 39200) loss: 2.608488
(Iteration 6651 / 39200) loss: 2.766192
(Iteration 6661 / 39200) loss: 2.726622
(Iteration 6671 / 39200) loss: 2.654963
(Iteration 6681 / 39200) loss: 3.201957
(Iteration 6691 / 39200) loss: 3.118755
(Iteration 6701 / 39200) loss: 2.735678
(Iteration 6711 / 39200) loss: 2.788282
(Iteration 6721 / 39200) loss: 2.923667
(Iteration 6731 / 39200) loss: 2.511683
(Iteration 6741 / 39200) loss: 2.776128
(Iteration 6751 / 39200) loss: 2.642837
(Iteration 6761 / 39200) loss: 2.501670
(Iteration 6771 / 39200) loss: 2.685888
(Iteration 6781 / 39200) loss: 2.547022
(Iteration 6791 / 39200) loss: 2.585200
(Iteration 6801 / 39200) loss: 2.425105
(Iteration 6811 / 39200) loss: 2.451837
(Iteration 6821 / 39200) loss: 2.703963
(Iteration 6831 / 39200) loss: 2.786157
(Iteration 6841 / 39200) loss: 2.541015
(Iteration 6851 / 39200) loss: 2.839837
(Iteration 6861 / 39200) loss: 2.767763
(Iteration 6871 / 39200) loss: 2.774644
(Iteration 6881 / 39200) loss: 2.806908
(Iteration 6891 / 39200) loss: 2.527258
(Iteration 6901 / 39200) loss: 2.757967
(Iteration 6911 / 39200) loss: 2.704607
(Iteration 6921 / 39200) loss: 2.848930
(Iteration 6931 / 39200) loss: 2.627783
(Iteration 6941 / 39200) loss: 2.808755
(Iteration 6951 / 39200) loss: 2.417890
(Iteration 6961 / 39200) loss: 2.389070
(Iteration 6971 / 39200) loss: 2.756622
```

```
(Iteration 6981 / 39200) loss: 2.388130
(Iteration 6991 / 39200) loss: 2.314782
(Iteration 7001 / 39200) loss: 2.569664
(Iteration 7011 / 39200) loss: 2.602977
(Iteration 7021 / 39200) loss: 2.961293
(Iteration 7031 / 39200) loss: 2.558148
(Iteration 7041 / 39200) loss: 2.524754
(Iteration 7051 / 39200) loss: 2.780173
(Iteration 7061 / 39200) loss: 2.881791
(Iteration 7071 / 39200) loss: 2.790525
(Iteration 7081 / 39200) loss: 2.930885
(Iteration 7091 / 39200) loss: 2.840406
(Iteration 7101 / 39200) loss: 2.757295
(Iteration 7111 / 39200) loss: 2.922941
(Iteration 7121 / 39200) loss: 2.717965
(Iteration 7131 / 39200) loss: 2.651823
(Iteration 7141 / 39200) loss: 2.411026
(Iteration 7151 / 39200) loss: 3.319947
(Iteration 7161 / 39200) loss: 2.979051
(Iteration 7171 / 39200) loss: 3.005132
(Iteration 7181 / 39200) loss: 2.430719
(Iteration 7191 / 39200) loss: 2.461254
(Iteration 7201 / 39200) loss: 3.172701
(Iteration 7211 / 39200) loss: 2.323735
(Iteration 7221 / 39200) loss: 2.811566
(Iteration 7231 / 39200) loss: 2.666888
(Iteration 7241 / 39200) loss: 2.970090
(Iteration 7251 / 39200) loss: 2.947163
(Iteration 7261 / 39200) loss: 2.762619
(Iteration 7271 / 39200) loss: 2.573029
(Iteration 7281 / 39200) loss: 2.872561
(Iteration 7291 / 39200) loss: 2.737487
(Iteration 7301 / 39200) loss: 2.309260
(Iteration 7311 / 39200) loss: 2.830468
(Iteration 7321 / 39200) loss: 3.190480
(Iteration 7331 / 39200) loss: 2.935444
(Iteration 7341 / 39200) loss: 2.513730
(Iteration 7351 / 39200) loss: 2.840356
(Iteration 7361 / 39200) loss: 2.528645
(Iteration 7371 / 39200) loss: 2.814724
(Iteration 7381 / 39200) loss: 2.603926
(Iteration 7391 / 39200) loss: 2.570947
(Iteration 7401 / 39200) loss: 2.592800
(Iteration 7411 / 39200) loss: 2.766573
(Iteration 7421 / 39200) loss: 2.615826
(Iteration 7431 / 39200) loss: 2.462600
(Iteration 7441 / 39200) loss: 2.601031
(Iteration 7451 / 39200) loss: 3.037313
```

```
(Iteration 7461 / 39200) loss: 2.546619
(Iteration 7471 / 39200) loss: 2.686359
(Iteration 7481 / 39200) loss: 2.792761
(Iteration 7491 / 39200) loss: 2.752274
(Iteration 7501 / 39200) loss: 2.766967
(Iteration 7511 / 39200) loss: 2.644007
(Iteration 7521 / 39200) loss: 2.454083
(Iteration 7531 / 39200) loss: 2.355246
(Iteration 7541 / 39200) loss: 3.078180
(Iteration 7551 / 39200) loss: 2.612266
(Iteration 7561 / 39200) loss: 2.720474
(Iteration 7571 / 39200) loss: 2.499305
(Iteration 7581 / 39200) loss: 2.740044
(Iteration 7591 / 39200) loss: 2.625151
(Iteration 7601 / 39200) loss: 2.588654
(Iteration 7611 / 39200) loss: 2.749681
(Iteration 7621 / 39200) loss: 2.647290
(Iteration 7631 / 39200) loss: 2.521554
(Iteration 7641 / 39200) loss: 2.040884
(Iteration 7651 / 39200) loss: 2.890083
(Iteration 7661 / 39200) loss: 2.340039
(Iteration 7671 / 39200) loss: 2.569522
(Iteration 7681 / 39200) loss: 2.648333
(Iteration 7691 / 39200) loss: 2.818752
(Iteration 7701 / 39200) loss: 2.726957
(Iteration 7711 / 39200) loss: 2.553115
(Iteration 7721 / 39200) loss: 2.608506
(Iteration 7731 / 39200) loss: 2.695130
(Iteration 7741 / 39200) loss: 2.312658
(Iteration 7751 / 39200) loss: 2.480107
(Iteration 7761 / 39200) loss: 2.422658
(Iteration 7771 / 39200) loss: 2.569550
(Iteration 7781 / 39200) loss: 3.167758
(Iteration 7791 / 39200) loss: 2.917205
(Iteration 7801 / 39200) loss: 2.597577
(Iteration 7811 / 39200) loss: 2.528388
(Iteration 7821 / 39200) loss: 2.806816
(Iteration 7831 / 39200) loss: 2.453023
(Epoch 4 / 20) train acc: 0.462000; val_acc: 0.433000
(Iteration 7841 / 39200) loss: 2.858376
(Iteration 7851 / 39200) loss: 2.589717
(Iteration 7861 / 39200) loss: 2.769856
(Iteration 7871 / 39200) loss: 2.803951
(Iteration 7881 / 39200) loss: 2.677814
(Iteration 7891 / 39200) loss: 2.293980
(Iteration 7901 / 39200) loss: 2.698806
(Iteration 7911 / 39200) loss: 3.148886
(Iteration 7921 / 39200) loss: 2.854247
```

```
(Iteration 7931 / 39200) loss: 3.079928
(Iteration 7941 / 39200) loss: 2.690137
(Iteration 7951 / 39200) loss: 2.693133
(Iteration 7961 / 39200) loss: 3.094355
(Iteration 7971 / 39200) loss: 2.639967
(Iteration 7981 / 39200) loss: 2.577032
(Iteration 7991 / 39200) loss: 2.401754
(Iteration 8001 / 39200) loss: 2.594490
(Iteration 8011 / 39200) loss: 2.168834
(Iteration 8021 / 39200) loss: 2.759346
(Iteration 8031 / 39200) loss: 2.935658
(Iteration 8041 / 39200) loss: 2.417450
(Iteration 8051 / 39200) loss: 2.609995
(Iteration 8061 / 39200) loss: 2.483052
(Iteration 8071 / 39200) loss: 2.438614
(Iteration 8081 / 39200) loss: 2.764725
(Iteration 8091 / 39200) loss: 2.208337
(Iteration 8101 / 39200) loss: 2.436235
(Iteration 8111 / 39200) loss: 2.846429
(Iteration 8121 / 39200) loss: 2.467129
(Iteration 8131 / 39200) loss: 2.459679
(Iteration 8141 / 39200) loss: 2.646611
(Iteration 8151 / 39200) loss: 2.923657
(Iteration 8161 / 39200) loss: 2.429350
(Iteration 8171 / 39200) loss: 2.507208
(Iteration 8181 / 39200) loss: 2.738965
(Iteration 8191 / 39200) loss: 2.854908
(Iteration 8201 / 39200) loss: 2.622550
(Iteration 8211 / 39200) loss: 2.506714
(Iteration 8221 / 39200) loss: 2.389472
(Iteration 8231 / 39200) loss: 2.558545
(Iteration 8241 / 39200) loss: 2.071529
(Iteration 8251 / 39200) loss: 2.534921
(Iteration 8261 / 39200) loss: 2.533434
(Iteration 8271 / 39200) loss: 2.146236
(Iteration 8281 / 39200) loss: 2.665887
(Iteration 8291 / 39200) loss: 2.394823
(Iteration 8301 / 39200) loss: 2.441216
(Iteration 8311 / 39200) loss: 3.033319
(Iteration 8321 / 39200) loss: 2.388021
(Iteration 8331 / 39200) loss: 2.685862
(Iteration 8341 / 39200) loss: 2.409812
(Iteration 8351 / 39200) loss: 2.396900
(Iteration 8361 / 39200) loss: 2.756927
(Iteration 8371 / 39200) loss: 2.610897
(Iteration 8381 / 39200) loss: 2.490866
(Iteration 8391 / 39200) loss: 2.576034
(Iteration 8401 / 39200) loss: 2.802002
```

```
(Iteration 8411 / 39200) loss: 2.634122
(Iteration 8421 / 39200) loss: 2.392447
(Iteration 8431 / 39200) loss: 2.503989
(Iteration 8441 / 39200) loss: 2.263601
(Iteration 8451 / 39200) loss: 3.092306
(Iteration 8461 / 39200) loss: 2.376874
(Iteration 8471 / 39200) loss: 2.577077
(Iteration 8481 / 39200) loss: 2.570697
(Iteration 8491 / 39200) loss: 2.405280
(Iteration 8501 / 39200) loss: 2.574083
(Iteration 8511 / 39200) loss: 2.789279
(Iteration 8521 / 39200) loss: 2.770455
(Iteration 8531 / 39200) loss: 2.520671
(Iteration 8541 / 39200) loss: 2.784240
(Iteration 8551 / 39200) loss: 2.519849
(Iteration 8561 / 39200) loss: 2.332082
(Iteration 8571 / 39200) loss: 2.650893
(Iteration 8581 / 39200) loss: 2.847083
(Iteration 8591 / 39200) loss: 2.460545
(Iteration 8601 / 39200) loss: 2.382474
(Iteration 8611 / 39200) loss: 2.295023
(Iteration 8621 / 39200) loss: 3.044360
(Iteration 8631 / 39200) loss: 2.520005
(Iteration 8641 / 39200) loss: 2.773758
(Iteration 8651 / 39200) loss: 2.381872
(Iteration 8661 / 39200) loss: 2.258105
(Iteration 8671 / 39200) loss: 2.542874
(Iteration 8681 / 39200) loss: 2.393966
(Iteration 8691 / 39200) loss: 2.294847
(Iteration 8701 / 39200) loss: 2.854615
(Iteration 8711 / 39200) loss: 2.019979
(Iteration 8721 / 39200) loss: 2.506673
(Iteration 8731 / 39200) loss: 2.537309
(Iteration 8741 / 39200) loss: 2.125317
(Iteration 8751 / 39200) loss: 2.363166
(Iteration 8761 / 39200) loss: 2.469737
(Iteration 8771 / 39200) loss: 2.371316
(Iteration 8781 / 39200) loss: 2.379391
(Iteration 8791 / 39200) loss: 2.628635
(Iteration 8801 / 39200) loss: 2.314391
(Iteration 8811 / 39200) loss: 2.281583
(Iteration 8821 / 39200) loss: 2.493992
(Iteration 8831 / 39200) loss: 2.615143
(Iteration 8841 / 39200) loss: 2.681210
(Iteration 8851 / 39200) loss: 2.474009
(Iteration 8861 / 39200) loss: 2.411096
(Iteration 8871 / 39200) loss: 2.383448
(Iteration 8881 / 39200) loss: 2.378575
```

```
(Iteration 8891 / 39200) loss: 2.345836
(Iteration 8901 / 39200) loss: 2.479496
(Iteration 8911 / 39200) loss: 2.386145
(Iteration 8921 / 39200) loss: 2.367099
(Iteration 8931 / 39200) loss: 2.306058
(Iteration 8941 / 39200) loss: 2.854540
(Iteration 8951 / 39200) loss: 2.532938
(Iteration 8961 / 39200) loss: 2.230452
(Iteration 8971 / 39200) loss: 2.675864
(Iteration 8981 / 39200) loss: 2.194485
(Iteration 8991 / 39200) loss: 2.156224
(Iteration 9001 / 39200) loss: 2.326844
(Iteration 9011 / 39200) loss: 2.637640
(Iteration 9021 / 39200) loss: 2.195439
(Iteration 9031 / 39200) loss: 2.661196
(Iteration 9041 / 39200) loss: 2.525357
(Iteration 9051 / 39200) loss: 2.140388
(Iteration 9061 / 39200) loss: 2.449395
(Iteration 9071 / 39200) loss: 2.084010
(Iteration 9081 / 39200) loss: 2.586638
(Iteration 9091 / 39200) loss: 2.489156
(Iteration 9101 / 39200) loss: 2.444052
(Iteration 9111 / 39200) loss: 2.443628
(Iteration 9121 / 39200) loss: 2.478374
(Iteration 9131 / 39200) loss: 2.608095
(Iteration 9141 / 39200) loss: 2.194953
(Iteration 9151 / 39200) loss: 2.646163
(Iteration 9161 / 39200) loss: 2.128305
(Iteration 9171 / 39200) loss: 2.279455
(Iteration 9181 / 39200) loss: 2.246380
(Iteration 9191 / 39200) loss: 2.589769
(Iteration 9201 / 39200) loss: 2.514722
(Iteration 9211 / 39200) loss: 2.325065
(Iteration 9221 / 39200) loss: 2.935788
(Iteration 9231 / 39200) loss: 2.374931
(Iteration 9241 / 39200) loss: 2.384379
(Iteration 9251 / 39200) loss: 2.723201
(Iteration 9261 / 39200) loss: 2.773832
(Iteration 9271 / 39200) loss: 2.568688
(Iteration 9281 / 39200) loss: 2.099143
(Iteration 9291 / 39200) loss: 2.223575
(Iteration 9301 / 39200) loss: 2.163885
(Iteration 9311 / 39200) loss: 2.730913
(Iteration 9321 / 39200) loss: 2.492015
(Iteration 9331 / 39200) loss: 2.251436
(Iteration 9341 / 39200) loss: 2.234956
(Iteration 9351 / 39200) loss: 2.629775
(Iteration 9361 / 39200) loss: 2.185168
```

```
(Iteration 9371 / 39200) loss: 2.605225
(Iteration 9381 / 39200) loss: 2.625117
(Iteration 9391 / 39200) loss: 2.353420
(Iteration 9401 / 39200) loss: 2.293086
(Iteration 9411 / 39200) loss: 2.006072
(Iteration 9421 / 39200) loss: 2.462793
(Iteration 9431 / 39200) loss: 2.320225
(Iteration 9441 / 39200) loss: 2.594384
(Iteration 9451 / 39200) loss: 2.507551
(Iteration 9461 / 39200) loss: 2.605693
(Iteration 9471 / 39200) loss: 2.527952
(Iteration 9481 / 39200) loss: 2.808211
(Iteration 9491 / 39200) loss: 2.682349
(Iteration 9501 / 39200) loss: 2.184558
(Iteration 9511 / 39200) loss: 2.468501
(Iteration 9521 / 39200) loss: 2.516015
(Iteration 9531 / 39200) loss: 2.496971
(Iteration 9541 / 39200) loss: 2.436392
(Iteration 9551 / 39200) loss: 2.323815
(Iteration 9561 / 39200) loss: 2.468712
(Iteration 9571 / 39200) loss: 2.239577
(Iteration 9581 / 39200) loss: 2.356727
(Iteration 9591 / 39200) loss: 2.614583
(Iteration 9601 / 39200) loss: 2.338272
(Iteration 9611 / 39200) loss: 2.581691
(Iteration 9621 / 39200) loss: 2.006143
(Iteration 9631 / 39200) loss: 2.139523
(Iteration 9641 / 39200) loss: 2.126961
(Iteration 9651 / 39200) loss: 2.395174
(Iteration 9661 / 39200) loss: 2.256902
(Iteration 9671 / 39200) loss: 2.308751
(Iteration 9681 / 39200) loss: 2.606837
(Iteration 9691 / 39200) loss: 2.973203
(Iteration 9701 / 39200) loss: 2.470260
(Iteration 9711 / 39200) loss: 2.381363
(Iteration 9721 / 39200) loss: 2.905298
(Iteration 9731 / 39200) loss: 2.443212
(Iteration 9741 / 39200) loss: 2.752347
(Iteration 9751 / 39200) loss: 2.451491
(Iteration 9761 / 39200) loss: 2.169401
(Iteration 9771 / 39200) loss: 2.397150
(Iteration 9781 / 39200) loss: 2.416705
(Iteration 9791 / 39200) loss: 2.460471
(Epoch 5 / 20) train acc: 0.503000; val_acc: 0.461000
(Iteration 9801 / 39200) loss: 2.264627
(Iteration 9811 / 39200) loss: 2.586425
(Iteration 9821 / 39200) loss: 2.166443
(Iteration 9831 / 39200) loss: 2.721601
```

```
(Iteration 9841 / 39200) loss: 2.489920
(Iteration 9851 / 39200) loss: 2.409428
(Iteration 9861 / 39200) loss: 2.319112
(Iteration 9871 / 39200) loss: 2.746194
(Iteration 9881 / 39200) loss: 3.023411
(Iteration 9891 / 39200) loss: 2.135071
(Iteration 9901 / 39200) loss: 2.636607
(Iteration 9911 / 39200) loss: 2.025764
(Iteration 9921 / 39200) loss: 2.497865
(Iteration 9931 / 39200) loss: 2.301787
(Iteration 9941 / 39200) loss: 2.809813
(Iteration 9951 / 39200) loss: 2.545678
(Iteration 9961 / 39200) loss: 2.330117
(Iteration 9971 / 39200) loss: 2.802582
(Iteration 9981 / 39200) loss: 2.692444
(Iteration 9991 / 39200) loss: 2.011875
(Iteration 10001 / 39200) loss: 2.464301
(Iteration 10011 / 39200) loss: 2.284230
(Iteration 10021 / 39200) loss: 2.743859
(Iteration 10031 / 39200) loss: 2.480980
(Iteration 10041 / 39200) loss: 2.557257
(Iteration 10051 / 39200) loss: 2.495068
(Iteration 10061 / 39200) loss: 2.444892
(Iteration 10071 / 39200) loss: 2.539230
(Iteration 10081 / 39200) loss: 2.756471
(Iteration 10091 / 39200) loss: 2.234706
(Iteration 10101 / 39200) loss: 2.043727
(Iteration 10111 / 39200) loss: 2.346061
(Iteration 10121 / 39200) loss: 2.481891
(Iteration 10131 / 39200) loss: 2.466571
(Iteration 10141 / 39200) loss: 2.558469
(Iteration 10151 / 39200) loss: 2.385873
(Iteration 10161 / 39200) loss: 2.457373
(Iteration 10171 / 39200) loss: 2.068666
(Iteration 10181 / 39200) loss: 2.471681
(Iteration 10191 / 39200) loss: 2.461428
(Iteration 10201 / 39200) loss: 2.279325
(Iteration 10211 / 39200) loss: 2.896808
(Iteration 10221 / 39200) loss: 2.270970
(Iteration 10231 / 39200) loss: 2.314899
(Iteration 10241 / 39200) loss: 2.372415
(Iteration 10251 / 39200) loss: 2.266357
(Iteration 10261 / 39200) loss: 2.469770
(Iteration 10271 / 39200) loss: 2.419653
(Iteration 10281 / 39200) loss: 2.253748
(Iteration 10291 / 39200) loss: 2.115682
(Iteration 10301 / 39200) loss: 2.952791
(Iteration 10311 / 39200) loss: 2.277229
```

```
(Iteration 10321 / 39200) loss: 2.476884
(Iteration 10331 / 39200) loss: 2.365570
(Iteration 10341 / 39200) loss: 2.614813
(Iteration 10351 / 39200) loss: 2.630120
(Iteration 10361 / 39200) loss: 2.087103
(Iteration 10371 / 39200) loss: 2.313790
(Iteration 10381 / 39200) loss: 2.700243
(Iteration 10391 / 39200) loss: 2.768360
(Iteration 10401 / 39200) loss: 2.418529
(Iteration 10411 / 39200) loss: 2.565372
(Iteration 10421 / 39200) loss: 2.455266
(Iteration 10431 / 39200) loss: 2.139935
(Iteration 10441 / 39200) loss: 2.202465
(Iteration 10451 / 39200) loss: 2.513422
(Iteration 10461 / 39200) loss: 2.631627
(Iteration 10471 / 39200) loss: 2.290942
(Iteration 10481 / 39200) loss: 2.340036
(Iteration 10491 / 39200) loss: 2.377097
(Iteration 10501 / 39200) loss: 2.353572
(Iteration 10511 / 39200) loss: 2.645207
(Iteration 10521 / 39200) loss: 2.142867
(Iteration 10531 / 39200) loss: 2.678497
(Iteration 10541 / 39200) loss: 2.244564
(Iteration 10551 / 39200) loss: 2.446355
(Iteration 10561 / 39200) loss: 2.309845
(Iteration 10571 / 39200) loss: 1.933863
(Iteration 10581 / 39200) loss: 2.258919
(Iteration 10591 / 39200) loss: 2.357858
(Iteration 10601 / 39200) loss: 2.423920
(Iteration 10611 / 39200) loss: 2.204038
(Iteration 10621 / 39200) loss: 2.285011
(Iteration 10631 / 39200) loss: 3.019133
(Iteration 10641 / 39200) loss: 2.021065
(Iteration 10651 / 39200) loss: 2.737905
(Iteration 10661 / 39200) loss: 2.497658
(Iteration 10671 / 39200) loss: 2.469629
(Iteration 10681 / 39200) loss: 2.158248
(Iteration 10691 / 39200) loss: 2.479839
(Iteration 10701 / 39200) loss: 2.344214
(Iteration 10711 / 39200) loss: 2.440999
(Iteration 10721 / 39200) loss: 2.398754
(Iteration 10731 / 39200) loss: 2.757689
(Iteration 10741 / 39200) loss: 2.222697
(Iteration 10751 / 39200) loss: 2.361101
(Iteration 10761 / 39200) loss: 2.160587
(Iteration 10771 / 39200) loss: 2.572665
(Iteration 10781 / 39200) loss: 2.423441
(Iteration 10791 / 39200) loss: 2.787923
```

```
(Iteration 10801 / 39200) loss: 2.018583
(Iteration 10811 / 39200) loss: 2.223790
(Iteration 10821 / 39200) loss: 2.214059
(Iteration 10831 / 39200) loss: 2.177848
(Iteration 10841 / 39200) loss: 2.256488
(Iteration 10851 / 39200) loss: 2.203240
(Iteration 10861 / 39200) loss: 2.236319
(Iteration 10871 / 39200) loss: 2.442635
(Iteration 10881 / 39200) loss: 2.415630
(Iteration 10891 / 39200) loss: 1.998117
(Iteration 10901 / 39200) loss: 2.265504
(Iteration 10911 / 39200) loss: 2.507035
(Iteration 10921 / 39200) loss: 2.414269
(Iteration 10931 / 39200) loss: 2.381485
(Iteration 10941 / 39200) loss: 2.597314
(Iteration 10951 / 39200) loss: 2.431625
(Iteration 10961 / 39200) loss: 2.687139
(Iteration 10971 / 39200) loss: 2.277649
(Iteration 10981 / 39200) loss: 1.857621
(Iteration 10991 / 39200) loss: 2.389560
(Iteration 11001 / 39200) loss: 2.565596
(Iteration 11011 / 39200) loss: 2.305643
(Iteration 11021 / 39200) loss: 2.465719
(Iteration 11031 / 39200) loss: 2.198098
(Iteration 11041 / 39200) loss: 2.226223
(Iteration 11051 / 39200) loss: 2.182906
(Iteration 11061 / 39200) loss: 2.577882
(Iteration 11071 / 39200) loss: 2.541534
(Iteration 11081 / 39200) loss: 2.486726
(Iteration 11091 / 39200) loss: 2.459154
(Iteration 11101 / 39200) loss: 2.566056
(Iteration 11111 / 39200) loss: 2.672749
(Iteration 11121 / 39200) loss: 2.379150
(Iteration 11131 / 39200) loss: 2.398494
(Iteration 11141 / 39200) loss: 2.408979
(Iteration 11151 / 39200) loss: 2.145469
(Iteration 11161 / 39200) loss: 2.306387
(Iteration 11171 / 39200) loss: 2.115699
(Iteration 11181 / 39200) loss: 2.244369
(Iteration 11191 / 39200) loss: 2.440930
(Iteration 11201 / 39200) loss: 2.111359
(Iteration 11211 / 39200) loss: 2.285821
(Iteration 11221 / 39200) loss: 2.327372
(Iteration 11231 / 39200) loss: 2.888202
(Iteration 11241 / 39200) loss: 2.453896
(Iteration 11251 / 39200) loss: 2.431300
(Iteration 11261 / 39200) loss: 2.094346
(Iteration 11271 / 39200) loss: 2.039252
```

```
(Iteration 11281 / 39200) loss: 2.371270
(Iteration 11291 / 39200) loss: 2.393605
(Iteration 11301 / 39200) loss: 2.630368
(Iteration 11311 / 39200) loss: 1.942175
(Iteration 11321 / 39200) loss: 2.479625
(Iteration 11331 / 39200) loss: 2.025489
(Iteration 11341 / 39200) loss: 2.007896
(Iteration 11351 / 39200) loss: 2.162811
(Iteration 11361 / 39200) loss: 1.792207
(Iteration 11371 / 39200) loss: 3.192556
(Iteration 11381 / 39200) loss: 2.025963
(Iteration 11391 / 39200) loss: 2.102518
(Iteration 11401 / 39200) loss: 2.250044
(Iteration 11411 / 39200) loss: 2.058139
(Iteration 11421 / 39200) loss: 2.307895
(Iteration 11431 / 39200) loss: 2.337139
(Iteration 11441 / 39200) loss: 2.258062
(Iteration 11451 / 39200) loss: 2.458836
(Iteration 11461 / 39200) loss: 2.229630
(Iteration 11471 / 39200) loss: 2.140126
(Iteration 11481 / 39200) loss: 2.385979
(Iteration 11491 / 39200) loss: 2.550703
(Iteration 11501 / 39200) loss: 1.969295
(Iteration 11511 / 39200) loss: 2.195470
(Iteration 11521 / 39200) loss: 2.310023
(Iteration 11531 / 39200) loss: 2.427383
(Iteration 11541 / 39200) loss: 2.110914
(Iteration 11551 / 39200) loss: 1.974623
(Iteration 11561 / 39200) loss: 2.428171
(Iteration 11571 / 39200) loss: 2.867272
(Iteration 11581 / 39200) loss: 2.591719
(Iteration 11591 / 39200) loss: 2.184724
(Iteration 11601 / 39200) loss: 2.315553
(Iteration 11611 / 39200) loss: 2.227487
(Iteration 11621 / 39200) loss: 2.162753
(Iteration 11631 / 39200) loss: 2.449399
(Iteration 11641 / 39200) loss: 2.038067
(Iteration 11651 / 39200) loss: 2.034777
(Iteration 11661 / 39200) loss: 2.202123
(Iteration 11671 / 39200) loss: 2.554246
(Iteration 11681 / 39200) loss: 2.060603
(Iteration 11691 / 39200) loss: 2.368094
(Iteration 11701 / 39200) loss: 2.229421
(Iteration 11711 / 39200) loss: 2.788989
(Iteration 11721 / 39200) loss: 2.552016
(Iteration 11731 / 39200) loss: 1.904370
(Iteration 11741 / 39200) loss: 2.602146
(Iteration 11751 / 39200) loss: 1.879974
```

```
(Epoch 6 / 20) train acc: 0.538000; val_acc: 0.438000
(Iteration 11761 / 39200) loss: 1.979189
(Iteration 11771 / 39200) loss: 2.355720
(Iteration 11781 / 39200) loss: 2.142097
(Iteration 11791 / 39200) loss: 2.090224
(Iteration 11801 / 39200) loss: 2.282101
(Iteration 11811 / 39200) loss: 2.211361
(Iteration 11821 / 39200) loss: 2.759735
(Iteration 11831 / 39200) loss: 2.154543
(Iteration 11841 / 39200) loss: 2.349467
(Iteration 11851 / 39200) loss: 2.042894
(Iteration 11861 / 39200) loss: 1.970970
(Iteration 11871 / 39200) loss: 2.436329
(Iteration 11881 / 39200) loss: 2.477346
(Iteration 11891 / 39200) loss: 2.699930
(Iteration 11901 / 39200) loss: 2.470941
(Iteration 11911 / 39200) loss: 2.022253
(Iteration 11921 / 39200) loss: 1.825896
(Iteration 11931 / 39200) loss: 2.397308
(Iteration 11941 / 39200) loss: 2.137500
(Iteration 11951 / 39200) loss: 2.650414
(Iteration 11961 / 39200) loss: 2.058806
(Iteration 11971 / 39200) loss: 2.280272
(Iteration 11981 / 39200) loss: 2.520436
(Iteration 11991 / 39200) loss: 2.158540
(Iteration 12001 / 39200) loss: 2.266535
(Iteration 12011 / 39200) loss: 2.201291
(Iteration 12021 / 39200) loss: 2.771787
(Iteration 12031 / 39200) loss: 2.299965
(Iteration 12041 / 39200) loss: 2.309520
(Iteration 12051 / 39200) loss: 2.182461
(Iteration 12061 / 39200) loss: 1.865017
(Iteration 12071 / 39200) loss: 2.534531
(Iteration 12081 / 39200) loss: 2.530335
(Iteration 12091 / 39200) loss: 2.000599
(Iteration 12101 / 39200) loss: 2.316093
(Iteration 12111 / 39200) loss: 2.311310
(Iteration 12121 / 39200) loss: 1.937944
(Iteration 12131 / 39200) loss: 2.165120
(Iteration 12141 / 39200) loss: 2.281972
(Iteration 12151 / 39200) loss: 2.231606
(Iteration 12161 / 39200) loss: 2.369304
(Iteration 12171 / 39200) loss: 2.525726
(Iteration 12181 / 39200) loss: 1.952248
(Iteration 12191 / 39200) loss: 2.180369
(Iteration 12201 / 39200) loss: 1.993091
(Iteration 12211 / 39200) loss: 2.178996
(Iteration 12221 / 39200) loss: 2.817323
```

```
(Iteration 12231 / 39200) loss: 2.037727
(Iteration 12241 / 39200) loss: 2.091661
(Iteration 12251 / 39200) loss: 2.172773
(Iteration 12261 / 39200) loss: 2.112970
(Iteration 12271 / 39200) loss: 1.895816
(Iteration 12281 / 39200) loss: 2.349302
(Iteration 12291 / 39200) loss: 2.088147
(Iteration 12301 / 39200) loss: 2.484771
(Iteration 12311 / 39200) loss: 1.946167
(Iteration 12321 / 39200) loss: 2.286650
(Iteration 12331 / 39200) loss: 2.076562
(Iteration 12341 / 39200) loss: 2.295994
(Iteration 12351 / 39200) loss: 2.295768
(Iteration 12361 / 39200) loss: 2.449464
(Iteration 12371 / 39200) loss: 2.386622
(Iteration 12381 / 39200) loss: 2.118466
(Iteration 12391 / 39200) loss: 2.201452
(Iteration 12401 / 39200) loss: 2.256974
(Iteration 12411 / 39200) loss: 2.362837
(Iteration 12421 / 39200) loss: 2.445071
(Iteration 12431 / 39200) loss: 2.216831
(Iteration 12441 / 39200) loss: 2.361428
(Iteration 12451 / 39200) loss: 2.376032
(Iteration 12461 / 39200) loss: 2.151892
(Iteration 12471 / 39200) loss: 2.274250
(Iteration 12481 / 39200) loss: 2.364304
(Iteration 12491 / 39200) loss: 1.936816
(Iteration 12501 / 39200) loss: 2.126376
(Iteration 12511 / 39200) loss: 1.943787
(Iteration 12521 / 39200) loss: 2.227077
(Iteration 12531 / 39200) loss: 2.070872
(Iteration 12541 / 39200) loss: 2.093182
(Iteration 12551 / 39200) loss: 1.866390
(Iteration 12561 / 39200) loss: 2.229120
(Iteration 12571 / 39200) loss: 1.920212
(Iteration 12581 / 39200) loss: 2.113845
(Iteration 12591 / 39200) loss: 2.307684
(Iteration 12601 / 39200) loss: 2.168294
(Iteration 12611 / 39200) loss: 2.376391
(Iteration 12621 / 39200) loss: 2.040319
(Iteration 12631 / 39200) loss: 2.029042
(Iteration 12641 / 39200) loss: 2.065278
(Iteration 12651 / 39200) loss: 2.084353
(Iteration 12661 / 39200) loss: 2.509258
(Iteration 12671 / 39200) loss: 2.091600
(Iteration 12681 / 39200) loss: 2.018609
(Iteration 12691 / 39200) loss: 2.134872
(Iteration 12701 / 39200) loss: 1.998630
```

```
(Iteration 12711 / 39200) loss: 2.158276
(Iteration 12721 / 39200) loss: 2.551520
(Iteration 12731 / 39200) loss: 2.309421
(Iteration 12741 / 39200) loss: 2.079789
(Iteration 12751 / 39200) loss: 1.730804
(Iteration 12761 / 39200) loss: 2.159066
(Iteration 12771 / 39200) loss: 2.191267
(Iteration 12781 / 39200) loss: 2.738579
(Iteration 12791 / 39200) loss: 2.357520
(Iteration 12801 / 39200) loss: 2.042453
(Iteration 12811 / 39200) loss: 1.716202
(Iteration 12821 / 39200) loss: 2.556110
(Iteration 12831 / 39200) loss: 1.782908
(Iteration 12841 / 39200) loss: 1.933134
(Iteration 12851 / 39200) loss: 2.065686
(Iteration 12861 / 39200) loss: 2.378015
(Iteration 12871 / 39200) loss: 2.099024
(Iteration 12881 / 39200) loss: 2.120027
(Iteration 12891 / 39200) loss: 2.243454
(Iteration 12901 / 39200) loss: 2.184894
(Iteration 12911 / 39200) loss: 1.933836
(Iteration 12921 / 39200) loss: 2.391237
(Iteration 12931 / 39200) loss: 2.305503
(Iteration 12941 / 39200) loss: 2.302366
(Iteration 12951 / 39200) loss: 2.120565
(Iteration 12961 / 39200) loss: 2.413155
(Iteration 12971 / 39200) loss: 2.161821
(Iteration 12981 / 39200) loss: 2.171170
(Iteration 12991 / 39200) loss: 1.994306
(Iteration 13001 / 39200) loss: 1.973853
(Iteration 13011 / 39200) loss: 2.216966
(Iteration 13021 / 39200) loss: 2.518014
(Iteration 13031 / 39200) loss: 1.970904
(Iteration 13041 / 39200) loss: 2.077990
(Iteration 13051 / 39200) loss: 2.615079
(Iteration 13061 / 39200) loss: 1.892798
(Iteration 13071 / 39200) loss: 2.127845
(Iteration 13081 / 39200) loss: 2.529705
(Iteration 13091 / 39200) loss: 1.738469
(Iteration 13101 / 39200) loss: 2.198112
(Iteration 13111 / 39200) loss: 2.061549
(Iteration 13121 / 39200) loss: 2.226194
(Iteration 13131 / 39200) loss: 2.321732
(Iteration 13141 / 39200) loss: 2.174111
(Iteration 13151 / 39200) loss: 2.181544
(Iteration 13161 / 39200) loss: 2.405718
(Iteration 13171 / 39200) loss: 2.420305
(Iteration 13181 / 39200) loss: 2.226979
```

```
(Iteration 13191 / 39200) loss: 2.027375
(Iteration 13201 / 39200) loss: 2.309947
(Iteration 13211 / 39200) loss: 2.331695
(Iteration 13221 / 39200) loss: 2.260796
(Iteration 13231 / 39200) loss: 2.062307
(Iteration 13241 / 39200) loss: 2.253322
(Iteration 13251 / 39200) loss: 2.060796
(Iteration 13261 / 39200) loss: 2.304769
(Iteration 13271 / 39200) loss: 3.071222
(Iteration 13281 / 39200) loss: 1.859017
(Iteration 13291 / 39200) loss: 2.463246
(Iteration 13301 / 39200) loss: 2.183324
(Iteration 13311 / 39200) loss: 2.388452
(Iteration 13321 / 39200) loss: 1.885016
(Iteration 13331 / 39200) loss: 1.698159
(Iteration 13341 / 39200) loss: 2.299745
(Iteration 13351 / 39200) loss: 1.994349
(Iteration 13361 / 39200) loss: 2.878869
(Iteration 13371 / 39200) loss: 1.663981
(Iteration 13381 / 39200) loss: 2.163887
(Iteration 13391 / 39200) loss: 2.210189
(Iteration 13401 / 39200) loss: 2.439748
(Iteration 13411 / 39200) loss: 2.175664
(Iteration 13421 / 39200) loss: 2.047044
(Iteration 13431 / 39200) loss: 2.129566
(Iteration 13441 / 39200) loss: 2.061956
(Iteration 13451 / 39200) loss: 2.236466
(Iteration 13461 / 39200) loss: 1.978008
(Iteration 13471 / 39200) loss: 2.295625
(Iteration 13481 / 39200) loss: 2.172882
(Iteration 13491 / 39200) loss: 1.766116
(Iteration 13501 / 39200) loss: 2.000325
(Iteration 13511 / 39200) loss: 2.398733
(Iteration 13521 / 39200) loss: 2.624266
(Iteration 13531 / 39200) loss: 2.251828
(Iteration 13541 / 39200) loss: 2.058653
(Iteration 13551 / 39200) loss: 2.054643
(Iteration 13561 / 39200) loss: 2.117329
(Iteration 13571 / 39200) loss: 2.124958
(Iteration 13581 / 39200) loss: 2.247260
(Iteration 13591 / 39200) loss: 2.441395
(Iteration 13601 / 39200) loss: 1.964985
(Iteration 13611 / 39200) loss: 1.817582
(Iteration 13621 / 39200) loss: 2.899180
(Iteration 13631 / 39200) loss: 2.064432
(Iteration 13641 / 39200) loss: 2.051866
(Iteration 13651 / 39200) loss: 2.257135
(Iteration 13661 / 39200) loss: 1.998958
```

```
(Iteration 13671 / 39200) loss: 1.857665
(Iteration 13681 / 39200) loss: 2.092461
(Iteration 13691 / 39200) loss: 2.274501
(Iteration 13701 / 39200) loss: 1.965915
(Iteration 13711 / 39200) loss: 2.497527
(Epoch 7 / 20) train acc: 0.549000; val_acc: 0.458000
(Iteration 13721 / 39200) loss: 1.840967
(Iteration 13731 / 39200) loss: 1.857324
(Iteration 13741 / 39200) loss: 2.129981
(Iteration 13751 / 39200) loss: 1.762758
(Iteration 13761 / 39200) loss: 1.914001
(Iteration 13771 / 39200) loss: 1.866505
(Iteration 13781 / 39200) loss: 1.822753
(Iteration 13791 / 39200) loss: 2.274423
(Iteration 13801 / 39200) loss: 2.093492
(Iteration 13811 / 39200) loss: 2.105543
(Iteration 13821 / 39200) loss: 2.310783
(Iteration 13831 / 39200) loss: 2.145111
(Iteration 13841 / 39200) loss: 1.876727
(Iteration 13851 / 39200) loss: 2.099480
(Iteration 13861 / 39200) loss: 2.049827
(Iteration 13871 / 39200) loss: 2.020448
(Iteration 13881 / 39200) loss: 2.286454
(Iteration 13891 / 39200) loss: 2.185248
(Iteration 13901 / 39200) loss: 1.933527
(Iteration 13911 / 39200) loss: 2.209759
(Iteration 13921 / 39200) loss: 2.104487
(Iteration 13931 / 39200) loss: 2.264859
(Iteration 13941 / 39200) loss: 1.896632
(Iteration 13951 / 39200) loss: 2.194003
(Iteration 13961 / 39200) loss: 1.729744
(Iteration 13971 / 39200) loss: 2.148712
(Iteration 13981 / 39200) loss: 2.134817
(Iteration 13991 / 39200) loss: 2.242183
(Iteration 14001 / 39200) loss: 2.163657
(Iteration 14011 / 39200) loss: 1.847469
(Iteration 14021 / 39200) loss: 2.302201
(Iteration 14031 / 39200) loss: 2.209214
(Iteration 14041 / 39200) loss: 2.081059
(Iteration 14051 / 39200) loss: 2.413900
(Iteration 14061 / 39200) loss: 2.331172
(Iteration 14071 / 39200) loss: 2.033609
(Iteration 14081 / 39200) loss: 2.231666
(Iteration 14091 / 39200) loss: 2.280462
(Iteration 14101 / 39200) loss: 2.197034
(Iteration 14111 / 39200) loss: 2.077317
(Iteration 14121 / 39200) loss: 2.331736
(Iteration 14131 / 39200) loss: 1.775167
```

```
(Iteration 14141 / 39200) loss: 2.582098
(Iteration 14151 / 39200) loss: 1.951641
(Iteration 14161 / 39200) loss: 1.950087
(Iteration 14171 / 39200) loss: 1.798216
(Iteration 14181 / 39200) loss: 1.894187
(Iteration 14191 / 39200) loss: 1.835574
(Iteration 14201 / 39200) loss: 2.418487
(Iteration 14211 / 39200) loss: 2.621981
(Iteration 14221 / 39200) loss: 2.421222
(Iteration 14231 / 39200) loss: 1.715634
(Iteration 14241 / 39200) loss: 2.442275
(Iteration 14251 / 39200) loss: 1.572406
(Iteration 14261 / 39200) loss: 2.239605
(Iteration 14271 / 39200) loss: 1.953964
(Iteration 14281 / 39200) loss: 2.262474
(Iteration 14291 / 39200) loss: 2.201868
(Iteration 14301 / 39200) loss: 2.059696
(Iteration 14311 / 39200) loss: 2.102218
(Iteration 14321 / 39200) loss: 1.924635
(Iteration 14331 / 39200) loss: 1.940076
(Iteration 14341 / 39200) loss: 2.085074
(Iteration 14351 / 39200) loss: 1.654016
(Iteration 14361 / 39200) loss: 2.016208
(Iteration 14371 / 39200) loss: 2.386288
(Iteration 14381 / 39200) loss: 1.851660
(Iteration 14391 / 39200) loss: 2.094275
(Iteration 14401 / 39200) loss: 2.383916
(Iteration 14411 / 39200) loss: 2.188181
(Iteration 14421 / 39200) loss: 2.129651
(Iteration 14431 / 39200) loss: 2.096368
(Iteration 14441 / 39200) loss: 1.907504
(Iteration 14451 / 39200) loss: 2.151658
(Iteration 14461 / 39200) loss: 2.037508
(Iteration 14471 / 39200) loss: 1.807397
(Iteration 14481 / 39200) loss: 2.428918
(Iteration 14491 / 39200) loss: 1.997742
(Iteration 14501 / 39200) loss: 1.896710
(Iteration 14511 / 39200) loss: 2.458747
(Iteration 14521 / 39200) loss: 2.192805
(Iteration 14531 / 39200) loss: 2.264426
(Iteration 14541 / 39200) loss: 1.787931
(Iteration 14551 / 39200) loss: 1.934397
(Iteration 14561 / 39200) loss: 1.981827
(Iteration 14571 / 39200) loss: 1.800096
(Iteration 14581 / 39200) loss: 2.078691
(Iteration 14591 / 39200) loss: 2.255364
(Iteration 14601 / 39200) loss: 1.963491
(Iteration 14611 / 39200) loss: 1.899322
```

```
(Iteration 14621 / 39200) loss: 1.963150
(Iteration 14631 / 39200) loss: 1.909091
(Iteration 14641 / 39200) loss: 1.982452
(Iteration 14651 / 39200) loss: 1.826542
(Iteration 14661 / 39200) loss: 1.887357
(Iteration 14671 / 39200) loss: 2.513024
(Iteration 14681 / 39200) loss: 1.968981
(Iteration 14691 / 39200) loss: 1.738561
(Iteration 14701 / 39200) loss: 2.021460
(Iteration 14711 / 39200) loss: 2.337051
(Iteration 14721 / 39200) loss: 1.922113
(Iteration 14731 / 39200) loss: 2.029292
(Iteration 14741 / 39200) loss: 1.464858
(Iteration 14751 / 39200) loss: 1.602389
(Iteration 14761 / 39200) loss: 2.204480
(Iteration 14771 / 39200) loss: 2.154896
(Iteration 14781 / 39200) loss: 2.027952
(Iteration 14791 / 39200) loss: 1.946610
(Iteration 14801 / 39200) loss: 2.126842
(Iteration 14811 / 39200) loss: 2.520359
(Iteration 14821 / 39200) loss: 1.790673
(Iteration 14831 / 39200) loss: 1.774419
(Iteration 14841 / 39200) loss: 2.393311
(Iteration 14851 / 39200) loss: 2.219862
(Iteration 14861 / 39200) loss: 1.967527
(Iteration 14871 / 39200) loss: 2.138539
(Iteration 14881 / 39200) loss: 1.911730
(Iteration 14891 / 39200) loss: 2.019481
(Iteration 14901 / 39200) loss: 2.009053
(Iteration 14911 / 39200) loss: 2.308492
(Iteration 14921 / 39200) loss: 2.316918
(Iteration 14931 / 39200) loss: 1.917845
(Iteration 14941 / 39200) loss: 1.713252
(Iteration 14951 / 39200) loss: 2.312553
(Iteration 14961 / 39200) loss: 1.929709
(Iteration 14971 / 39200) loss: 1.784548
(Iteration 14981 / 39200) loss: 2.027081
(Iteration 14991 / 39200) loss: 1.975386
(Iteration 15001 / 39200) loss: 1.888967
(Iteration 15011 / 39200) loss: 2.032063
(Iteration 15021 / 39200) loss: 1.910258
(Iteration 15031 / 39200) loss: 2.076061
(Iteration 15041 / 39200) loss: 2.357645
(Iteration 15051 / 39200) loss: 2.038799
(Iteration 15061 / 39200) loss: 2.012789
(Iteration 15071 / 39200) loss: 2.014564
(Iteration 15081 / 39200) loss: 1.973443
(Iteration 15091 / 39200) loss: 1.711389
```

```
(Iteration 15101 / 39200) loss: 2.414490
(Iteration 15111 / 39200) loss: 1.922850
(Iteration 15121 / 39200) loss: 2.158607
(Iteration 15131 / 39200) loss: 2.285806
(Iteration 15141 / 39200) loss: 1.674621
(Iteration 15151 / 39200) loss: 2.031145
(Iteration 15161 / 39200) loss: 1.902786
(Iteration 15171 / 39200) loss: 2.177721
(Iteration 15181 / 39200) loss: 1.912710
(Iteration 15191 / 39200) loss: 1.870255
(Iteration 15201 / 39200) loss: 1.606156
(Iteration 15211 / 39200) loss: 2.239243
(Iteration 15221 / 39200) loss: 2.031728
(Iteration 15231 / 39200) loss: 2.223194
(Iteration 15241 / 39200) loss: 1.971686
(Iteration 15251 / 39200) loss: 1.961065
(Iteration 15261 / 39200) loss: 1.938650
(Iteration 15271 / 39200) loss: 1.926096
(Iteration 15281 / 39200) loss: 2.037803
(Iteration 15291 / 39200) loss: 1.531589
(Iteration 15301 / 39200) loss: 1.649912
(Iteration 15311 / 39200) loss: 2.008400
(Iteration 15321 / 39200) loss: 2.146187
(Iteration 15331 / 39200) loss: 1.868870
(Iteration 15341 / 39200) loss: 2.374446
(Iteration 15351 / 39200) loss: 1.881943
(Iteration 15361 / 39200) loss: 2.123700
(Iteration 15371 / 39200) loss: 1.532525
(Iteration 15381 / 39200) loss: 1.635450
(Iteration 15391 / 39200) loss: 2.066597
(Iteration 15401 / 39200) loss: 1.643647
(Iteration 15411 / 39200) loss: 1.899946
(Iteration 15421 / 39200) loss: 1.922949
(Iteration 15431 / 39200) loss: 1.722079
(Iteration 15441 / 39200) loss: 2.330344
(Iteration 15451 / 39200) loss: 2.166618
(Iteration 15461 / 39200) loss: 1.943003
(Iteration 15471 / 39200) loss: 1.932467
(Iteration 15481 / 39200) loss: 1.940919
(Iteration 15491 / 39200) loss: 1.865972
(Iteration 15501 / 39200) loss: 1.911424
(Iteration 15511 / 39200) loss: 2.096624
(Iteration 15521 / 39200) loss: 1.922434
(Iteration 15531 / 39200) loss: 1.712322
(Iteration 15541 / 39200) loss: 2.130674
(Iteration 15551 / 39200) loss: 1.767006
(Iteration 15561 / 39200) loss: 1.732551
(Iteration 15571 / 39200) loss: 2.781865
```

```
(Iteration 15581 / 39200) loss: 1.834946
(Iteration 15591 / 39200) loss: 2.165639
(Iteration 15601 / 39200) loss: 1.794280
(Iteration 15611 / 39200) loss: 1.724486
(Iteration 15621 / 39200) loss: 2.146421
(Iteration 15631 / 39200) loss: 1.536003
(Iteration 15641 / 39200) loss: 2.298498
(Iteration 15651 / 39200) loss: 2.020953
(Iteration 15661 / 39200) loss: 1.907071
(Iteration 15671 / 39200) loss: 2.032201
(Epoch 8 / 20) train acc: 0.556000; val_acc: 0.472000
(Iteration 15681 / 39200) loss: 2.035094
(Iteration 15691 / 39200) loss: 2.300423
(Iteration 15701 / 39200) loss: 2.049391
(Iteration 15711 / 39200) loss: 2.705320
(Iteration 15721 / 39200) loss: 1.810804
(Iteration 15731 / 39200) loss: 1.945690
(Iteration 15741 / 39200) loss: 1.854718
(Iteration 15751 / 39200) loss: 2.241599
(Iteration 15761 / 39200) loss: 1.845213
(Iteration 15771 / 39200) loss: 1.505875
(Iteration 15781 / 39200) loss: 2.567392
(Iteration 15791 / 39200) loss: 1.562495
(Iteration 15801 / 39200) loss: 2.066930
(Iteration 15811 / 39200) loss: 2.073472
(Iteration 15821 / 39200) loss: 2.115254
(Iteration 15831 / 39200) loss: 2.258504
(Iteration 15841 / 39200) loss: 2.224495
(Iteration 15851 / 39200) loss: 2.413635
(Iteration 15861 / 39200) loss: 1.793147
(Iteration 15871 / 39200) loss: 1.921304
(Iteration 15881 / 39200) loss: 1.891292
(Iteration 15891 / 39200) loss: 1.877419
(Iteration 15901 / 39200) loss: 2.317229
(Iteration 15911 / 39200) loss: 2.013321
(Iteration 15921 / 39200) loss: 2.058377
(Iteration 15931 / 39200) loss: 1.977227
(Iteration 15941 / 39200) loss: 1.958410
(Iteration 15951 / 39200) loss: 1.645216
(Iteration 15961 / 39200) loss: 2.020606
(Iteration 15971 / 39200) loss: 1.781201
(Iteration 15981 / 39200) loss: 2.126560
(Iteration 15991 / 39200) loss: 2.371099
(Iteration 16001 / 39200) loss: 2.068367
(Iteration 16011 / 39200) loss: 1.833479
(Iteration 16021 / 39200) loss: 1.877727
(Iteration 16031 / 39200) loss: 1.969797
(Iteration 16041 / 39200) loss: 1.965313
```

```
(Iteration 16051 / 39200) loss: 1.763288
(Iteration 16061 / 39200) loss: 1.991038
(Iteration 16071 / 39200) loss: 1.825028
(Iteration 16081 / 39200) loss: 1.789138
(Iteration 16091 / 39200) loss: 1.604213
(Iteration 16101 / 39200) loss: 2.179525
(Iteration 16111 / 39200) loss: 1.834411
(Iteration 16121 / 39200) loss: 1.630041
(Iteration 16131 / 39200) loss: 1.802056
(Iteration 16141 / 39200) loss: 2.078887
(Iteration 16151 / 39200) loss: 1.597876
(Iteration 16161 / 39200) loss: 2.039752
(Iteration 16171 / 39200) loss: 2.051152
(Iteration 16181 / 39200) loss: 1.740761
(Iteration 16191 / 39200) loss: 2.113307
(Iteration 16201 / 39200) loss: 2.015204
(Iteration 16211 / 39200) loss: 2.469800
(Iteration 16221 / 39200) loss: 1.763610
(Iteration 16231 / 39200) loss: 2.155225
(Iteration 16241 / 39200) loss: 1.672082
(Iteration 16251 / 39200) loss: 2.490596
(Iteration 16261 / 39200) loss: 1.884584
(Iteration 16271 / 39200) loss: 1.993025
(Iteration 16281 / 39200) loss: 1.805109
(Iteration 16291 / 39200) loss: 1.986492
(Iteration 16301 / 39200) loss: 2.481902
(Iteration 16311 / 39200) loss: 2.279050
(Iteration 16321 / 39200) loss: 1.818628
(Iteration 16331 / 39200) loss: 1.882120
(Iteration 16341 / 39200) loss: 1.742753
(Iteration 16351 / 39200) loss: 2.028008
(Iteration 16361 / 39200) loss: 1.951185
(Iteration 16371 / 39200) loss: 2.175285
(Iteration 16381 / 39200) loss: 2.215265
(Iteration 16391 / 39200) loss: 2.218504
(Iteration 16401 / 39200) loss: 1.863601
(Iteration 16411 / 39200) loss: 2.177160
(Iteration 16421 / 39200) loss: 2.081552
(Iteration 16431 / 39200) loss: 1.692278
(Iteration 16441 / 39200) loss: 1.870286
(Iteration 16451 / 39200) loss: 1.869560
(Iteration 16461 / 39200) loss: 2.221020
(Iteration 16471 / 39200) loss: 1.846401
(Iteration 16481 / 39200) loss: 2.187852
(Iteration 16491 / 39200) loss: 1.926110
(Iteration 16501 / 39200) loss: 2.343982
(Iteration 16511 / 39200) loss: 1.820249
(Iteration 16521 / 39200) loss: 1.985725
```

```
(Iteration 16531 / 39200) loss: 2.122870
(Iteration 16541 / 39200) loss: 1.987202
(Iteration 16551 / 39200) loss: 2.046702
(Iteration 16561 / 39200) loss: 1.894007
(Iteration 16571 / 39200) loss: 1.924518
(Iteration 16581 / 39200) loss: 1.873482
(Iteration 16591 / 39200) loss: 1.793706
(Iteration 16601 / 39200) loss: 1.689782
(Iteration 16611 / 39200) loss: 1.543275
(Iteration 16621 / 39200) loss: 1.735525
(Iteration 16631 / 39200) loss: 2.030345
(Iteration 16641 / 39200) loss: 1.781987
(Iteration 16651 / 39200) loss: 1.774353
(Iteration 16661 / 39200) loss: 1.995044
(Iteration 16671 / 39200) loss: 2.329973
(Iteration 16681 / 39200) loss: 2.203363
(Iteration 16691 / 39200) loss: 1.963582
(Iteration 16701 / 39200) loss: 2.288259
(Iteration 16711 / 39200) loss: 1.898305
(Iteration 16721 / 39200) loss: 2.101600
(Iteration 16731 / 39200) loss: 1.972216
(Iteration 16741 / 39200) loss: 1.655112
(Iteration 16751 / 39200) loss: 2.088269
(Iteration 16761 / 39200) loss: 2.140594
(Iteration 16771 / 39200) loss: 1.694736
(Iteration 16781 / 39200) loss: 1.838890
(Iteration 16791 / 39200) loss: 1.943010
(Iteration 16801 / 39200) loss: 2.168821
(Iteration 16811 / 39200) loss: 2.175992
(Iteration 16821 / 39200) loss: 1.729299
(Iteration 16831 / 39200) loss: 1.909784
(Iteration 16841 / 39200) loss: 1.821178
(Iteration 16851 / 39200) loss: 1.950804
(Iteration 16861 / 39200) loss: 1.673016
(Iteration 16871 / 39200) loss: 1.521359
(Iteration 16881 / 39200) loss: 1.999747
(Iteration 16891 / 39200) loss: 1.758990
(Iteration 16901 / 39200) loss: 1.904260
(Iteration 16911 / 39200) loss: 2.081953
(Iteration 16921 / 39200) loss: 2.173529
(Iteration 16931 / 39200) loss: 2.195571
(Iteration 16941 / 39200) loss: 1.918534
(Iteration 16951 / 39200) loss: 2.181572
(Iteration 16961 / 39200) loss: 2.422836
(Iteration 16971 / 39200) loss: 2.047293
(Iteration 16981 / 39200) loss: 2.033470
(Iteration 16991 / 39200) loss: 2.285945
(Iteration 17001 / 39200) loss: 1.926369
```

```
(Iteration 17011 / 39200) loss: 1.845999
(Iteration 17021 / 39200) loss: 1.862051
(Iteration 17031 / 39200) loss: 2.204894
(Iteration 17041 / 39200) loss: 1.647755
(Iteration 17051 / 39200) loss: 1.447497
(Iteration 17061 / 39200) loss: 1.899532
(Iteration 17071 / 39200) loss: 1.973766
(Iteration 17081 / 39200) loss: 1.925738
(Iteration 17091 / 39200) loss: 2.245525
(Iteration 17101 / 39200) loss: 2.025735
(Iteration 17111 / 39200) loss: 1.998355
(Iteration 17121 / 39200) loss: 1.967021
(Iteration 17131 / 39200) loss: 1.732386
(Iteration 17141 / 39200) loss: 1.493112
(Iteration 17151 / 39200) loss: 2.143447
(Iteration 17161 / 39200) loss: 2.144929
(Iteration 17171 / 39200) loss: 1.780934
(Iteration 17181 / 39200) loss: 1.954514
(Iteration 17191 / 39200) loss: 1.864817
(Iteration 17201 / 39200) loss: 2.268183
(Iteration 17211 / 39200) loss: 2.417037
(Iteration 17221 / 39200) loss: 1.422762
(Iteration 17231 / 39200) loss: 1.889895
(Iteration 17241 / 39200) loss: 1.613807
(Iteration 17251 / 39200) loss: 2.100242
(Iteration 17261 / 39200) loss: 2.069201
(Iteration 17271 / 39200) loss: 1.780317
(Iteration 17281 / 39200) loss: 1.879429
(Iteration 17291 / 39200) loss: 2.174324
(Iteration 17301 / 39200) loss: 1.936820
(Iteration 17311 / 39200) loss: 2.172552
(Iteration 17321 / 39200) loss: 1.538834
(Iteration 17331 / 39200) loss: 1.980652
(Iteration 17341 / 39200) loss: 1.885301
(Iteration 17351 / 39200) loss: 1.742534
(Iteration 17361 / 39200) loss: 1.916309
(Iteration 17371 / 39200) loss: 1.682471
(Iteration 17381 / 39200) loss: 2.045950
(Iteration 17391 / 39200) loss: 1.695929
(Iteration 17401 / 39200) loss: 2.166610
(Iteration 17411 / 39200) loss: 1.762628
(Iteration 17421 / 39200) loss: 1.673055
(Iteration 17431 / 39200) loss: 2.125930
(Iteration 17441 / 39200) loss: 1.989702
(Iteration 17451 / 39200) loss: 2.046822
(Iteration 17461 / 39200) loss: 2.171606
(Iteration 17471 / 39200) loss: 1.426524
(Iteration 17481 / 39200) loss: 2.319027
```

```
(Iteration 17491 / 39200) loss: 1.889803
(Iteration 17501 / 39200) loss: 1.824423
(Iteration 17511 / 39200) loss: 1.774019
(Iteration 17521 / 39200) loss: 1.949504
(Iteration 17531 / 39200) loss: 1.773816
(Iteration 17541 / 39200) loss: 2.094523
(Iteration 17551 / 39200) loss: 1.664585
(Iteration 17561 / 39200) loss: 1.373252
(Iteration 17571 / 39200) loss: 1.715920
(Iteration 17581 / 39200) loss: 2.015097
(Iteration 17591 / 39200) loss: 1.799845
(Iteration 17601 / 39200) loss: 2.256300
(Iteration 17611 / 39200) loss: 2.028014
(Iteration 17621 / 39200) loss: 1.903263
(Iteration 17631 / 39200) loss: 1.719129
(Epoch 9 / 20) train acc: 0.596000; val_acc: 0.461000
(Iteration 17641 / 39200) loss: 1.668902
(Iteration 17651 / 39200) loss: 1.995853
(Iteration 17661 / 39200) loss: 2.064407
(Iteration 17671 / 39200) loss: 1.952896
(Iteration 17681 / 39200) loss: 1.861689
(Iteration 17691 / 39200) loss: 1.950541
(Iteration 17701 / 39200) loss: 2.199399
(Iteration 17711 / 39200) loss: 2.075653
(Iteration 17721 / 39200) loss: 1.515916
(Iteration 17731 / 39200) loss: 1.701986
(Iteration 17741 / 39200) loss: 1.938326
(Iteration 17751 / 39200) loss: 1.680133
(Iteration 17761 / 39200) loss: 2.062005
(Iteration 17771 / 39200) loss: 2.331702
(Iteration 17781 / 39200) loss: 1.612079
(Iteration 17791 / 39200) loss: 2.561366
(Iteration 17801 / 39200) loss: 2.021608
(Iteration 17811 / 39200) loss: 1.959825
(Iteration 17821 / 39200) loss: 1.886561
(Iteration 17831 / 39200) loss: 2.090073
(Iteration 17841 / 39200) loss: 1.941055
(Iteration 17851 / 39200) loss: 1.528340
(Iteration 17861 / 39200) loss: 2.221514
(Iteration 17871 / 39200) loss: 1.817259
(Iteration 17881 / 39200) loss: 1.941885
(Iteration 17891 / 39200) loss: 1.782970
(Iteration 17901 / 39200) loss: 1.691485
(Iteration 17911 / 39200) loss: 1.788861
(Iteration 17921 / 39200) loss: 1.945656
(Iteration 17931 / 39200) loss: 1.837478
(Iteration 17941 / 39200) loss: 1.828652
(Iteration 17951 / 39200) loss: 2.238521
```

```
(Iteration 17961 / 39200) loss: 1.674025
(Iteration 17971 / 39200) loss: 1.932964
(Iteration 17981 / 39200) loss: 1.756878
(Iteration 17991 / 39200) loss: 2.078699
(Iteration 18001 / 39200) loss: 1.707588
(Iteration 18011 / 39200) loss: 2.034408
(Iteration 18021 / 39200) loss: 1.815098
(Iteration 18031 / 39200) loss: 1.627053
(Iteration 18041 / 39200) loss: 1.712591
(Iteration 18051 / 39200) loss: 1.827736
(Iteration 18061 / 39200) loss: 1.599743
(Iteration 18071 / 39200) loss: 1.888593
(Iteration 18081 / 39200) loss: 1.814035
(Iteration 18091 / 39200) loss: 2.019677
(Iteration 18101 / 39200) loss: 1.753121
(Iteration 18111 / 39200) loss: 2.182326
(Iteration 18121 / 39200) loss: 1.730270
(Iteration 18131 / 39200) loss: 2.059759
(Iteration 18141 / 39200) loss: 1.586739
(Iteration 18151 / 39200) loss: 1.736072
(Iteration 18161 / 39200) loss: 1.878009
(Iteration 18171 / 39200) loss: 1.576408
(Iteration 18181 / 39200) loss: 1.690977
(Iteration 18191 / 39200) loss: 1.848247
(Iteration 18201 / 39200) loss: 1.785999
(Iteration 18211 / 39200) loss: 1.618807
(Iteration 18221 / 39200) loss: 1.430290
(Iteration 18231 / 39200) loss: 1.448583
(Iteration 18241 / 39200) loss: 1.498324
(Iteration 18251 / 39200) loss: 1.746545
(Iteration 18261 / 39200) loss: 2.225793
(Iteration 18271 / 39200) loss: 2.098356
(Iteration 18281 / 39200) loss: 1.922588
(Iteration 18291 / 39200) loss: 1.466304
(Iteration 18301 / 39200) loss: 1.990980
(Iteration 18311 / 39200) loss: 1.774654
(Iteration 18321 / 39200) loss: 2.168213
(Iteration 18331 / 39200) loss: 2.079892
(Iteration 18341 / 39200) loss: 2.144027
(Iteration 18351 / 39200) loss: 2.066364
(Iteration 18361 / 39200) loss: 1.387556
(Iteration 18371 / 39200) loss: 2.343798
(Iteration 18381 / 39200) loss: 1.836735
(Iteration 18391 / 39200) loss: 1.928148
(Iteration 18401 / 39200) loss: 2.228350
(Iteration 18411 / 39200) loss: 1.650221
(Iteration 18421 / 39200) loss: 1.430612
(Iteration 18431 / 39200) loss: 1.842326
```

```
(Iteration 18441 / 39200) loss: 1.767850
(Iteration 18451 / 39200) loss: 1.662929
(Iteration 18461 / 39200) loss: 1.642027
(Iteration 18471 / 39200) loss: 2.249773
(Iteration 18481 / 39200) loss: 1.994949
(Iteration 18491 / 39200) loss: 2.140814
(Iteration 18501 / 39200) loss: 1.918588
(Iteration 18511 / 39200) loss: 2.348246
(Iteration 18521 / 39200) loss: 2.140848
(Iteration 18531 / 39200) loss: 1.886179
(Iteration 18541 / 39200) loss: 1.841416
(Iteration 18551 / 39200) loss: 1.752438
(Iteration 18561 / 39200) loss: 1.759297
(Iteration 18571 / 39200) loss: 1.489626
(Iteration 18581 / 39200) loss: 1.846411
(Iteration 18591 / 39200) loss: 1.895931
(Iteration 18601 / 39200) loss: 1.911841
(Iteration 18611 / 39200) loss: 1.641147
(Iteration 18621 / 39200) loss: 2.024801
(Iteration 18631 / 39200) loss: 1.713900
(Iteration 18641 / 39200) loss: 2.243827
(Iteration 18651 / 39200) loss: 2.073805
(Iteration 18661 / 39200) loss: 1.618367
(Iteration 18671 / 39200) loss: 1.512408
(Iteration 18681 / 39200) loss: 1.714933
(Iteration 18691 / 39200) loss: 2.006248
(Iteration 18701 / 39200) loss: 1.988805
(Iteration 18711 / 39200) loss: 1.364869
(Iteration 18721 / 39200) loss: 1.540432
(Iteration 18731 / 39200) loss: 1.750206
(Iteration 18741 / 39200) loss: 1.705402
(Iteration 18751 / 39200) loss: 1.827750
(Iteration 18761 / 39200) loss: 1.895804
(Iteration 18771 / 39200) loss: 2.261400
(Iteration 18781 / 39200) loss: 2.370285
(Iteration 18791 / 39200) loss: 1.749382
(Iteration 18801 / 39200) loss: 1.683372
(Iteration 18811 / 39200) loss: 1.710526
(Iteration 18821 / 39200) loss: 2.211007
(Iteration 18831 / 39200) loss: 1.777574
(Iteration 18841 / 39200) loss: 1.896147
(Iteration 18851 / 39200) loss: 1.731209
(Iteration 18861 / 39200) loss: 2.706907
(Iteration 18871 / 39200) loss: 1.831950
(Iteration 18881 / 39200) loss: 2.268780
(Iteration 18891 / 39200) loss: 1.835226
(Iteration 18901 / 39200) loss: 1.705167
(Iteration 18911 / 39200) loss: 1.618035
```

```
(Iteration 18921 / 39200) loss: 1.506643
(Iteration 18931 / 39200) loss: 1.987124
(Iteration 18941 / 39200) loss: 1.706955
(Iteration 18951 / 39200) loss: 2.175071
(Iteration 18961 / 39200) loss: 1.972591
(Iteration 18971 / 39200) loss: 1.910773
(Iteration 18981 / 39200) loss: 2.118836
(Iteration 18991 / 39200) loss: 1.966110
(Iteration 19001 / 39200) loss: 1.817656
(Iteration 19011 / 39200) loss: 1.905313
(Iteration 19021 / 39200) loss: 1.474735
(Iteration 19031 / 39200) loss: 1.533591
(Iteration 19041 / 39200) loss: 1.948543
(Iteration 19051 / 39200) loss: 1.667108
(Iteration 19061 / 39200) loss: 2.083956
(Iteration 19071 / 39200) loss: 1.716555
(Iteration 19081 / 39200) loss: 1.906005
(Iteration 19091 / 39200) loss: 1.907606
(Iteration 19101 / 39200) loss: 1.893485
(Iteration 19111 / 39200) loss: 1.809973
(Iteration 19121 / 39200) loss: 1.964338
(Iteration 19131 / 39200) loss: 1.600454
(Iteration 19141 / 39200) loss: 2.157796
(Iteration 19151 / 39200) loss: 1.902519
(Iteration 19161 / 39200) loss: 1.736846
(Iteration 19171 / 39200) loss: 1.554460
(Iteration 19181 / 39200) loss: 1.673513
(Iteration 19191 / 39200) loss: 2.170277
(Iteration 19201 / 39200) loss: 1.750020
(Iteration 19211 / 39200) loss: 2.003398
(Iteration 19221 / 39200) loss: 1.756079
(Iteration 19231 / 39200) loss: 1.706582
(Iteration 19241 / 39200) loss: 1.879888
(Iteration 19251 / 39200) loss: 1.950808
(Iteration 19261 / 39200) loss: 2.194878
(Iteration 19271 / 39200) loss: 2.193540
(Iteration 19281 / 39200) loss: 2.125644
(Iteration 19291 / 39200) loss: 1.759929
(Iteration 19301 / 39200) loss: 1.925355
(Iteration 19311 / 39200) loss: 1.835936
(Iteration 19321 / 39200) loss: 2.156564
(Iteration 19331 / 39200) loss: 2.028158
(Iteration 19341 / 39200) loss: 1.399608
(Iteration 19351 / 39200) loss: 1.891182
(Iteration 19361 / 39200) loss: 1.836529
(Iteration 19371 / 39200) loss: 1.744092
(Iteration 19381 / 39200) loss: 1.670511
(Iteration 19391 / 39200) loss: 1.813757
```

```
(Iteration 19401 / 39200) loss: 1.861073
(Iteration 19411 / 39200) loss: 1.957110
(Iteration 19421 / 39200) loss: 1.621543
(Iteration 19431 / 39200) loss: 2.059084
(Iteration 19441 / 39200) loss: 1.974190
(Iteration 19451 / 39200) loss: 1.527373
(Iteration 19461 / 39200) loss: 1.780544
(Iteration 19471 / 39200) loss: 1.826672
(Iteration 19481 / 39200) loss: 1.612090
(Iteration 19491 / 39200) loss: 1.980154
(Iteration 19501 / 39200) loss: 1.423490
(Iteration 19511 / 39200) loss: 1.675886
(Iteration 19521 / 39200) loss: 1.684290
(Iteration 19531 / 39200) loss: 2.395890
(Iteration 19541 / 39200) loss: 1.577310
(Iteration 19551 / 39200) loss: 2.087548
(Iteration 19561 / 39200) loss: 1.683908
(Iteration 19571 / 39200) loss: 2.017737
(Iteration 19581 / 39200) loss: 2.166419
(Iteration 19591 / 39200) loss: 1.906493
(Epoch 10 / 20) train acc: 0.593000; val_acc: 0.482000
(Iteration 19601 / 39200) loss: 1.759444
(Iteration 19611 / 39200) loss: 1.433895
(Iteration 19621 / 39200) loss: 1.455365
(Iteration 19631 / 39200) loss: 1.739678
(Iteration 19641 / 39200) loss: 1.658748
(Iteration 19651 / 39200) loss: 1.991801
(Iteration 19661 / 39200) loss: 1.600676
(Iteration 19671 / 39200) loss: 2.063999
(Iteration 19681 / 39200) loss: 1.671524
(Iteration 19691 / 39200) loss: 1.867509
(Iteration 19701 / 39200) loss: 2.336072
(Iteration 19711 / 39200) loss: 1.643011
(Iteration 19721 / 39200) loss: 1.531766
(Iteration 19731 / 39200) loss: 1.683011
(Iteration 19741 / 39200) loss: 1.858837
(Iteration 19751 / 39200) loss: 1.809012
(Iteration 19761 / 39200) loss: 2.079158
(Iteration 19771 / 39200) loss: 1.625551
(Iteration 19781 / 39200) loss: 2.012864
(Iteration 19791 / 39200) loss: 1.765095
(Iteration 19801 / 39200) loss: 2.001919
(Iteration 19811 / 39200) loss: 1.562002
(Iteration 19821 / 39200) loss: 1.738772
(Iteration 19831 / 39200) loss: 2.150414
(Iteration 19841 / 39200) loss: 1.879884
(Iteration 19851 / 39200) loss: 1.874020
(Iteration 19861 / 39200) loss: 2.060339
```

```
(Iteration 19871 / 39200) loss: 1.773149
(Iteration 19881 / 39200) loss: 1.753751
(Iteration 19891 / 39200) loss: 1.502497
(Iteration 19901 / 39200) loss: 1.608576
(Iteration 19911 / 39200) loss: 1.814175
(Iteration 19921 / 39200) loss: 1.780970
(Iteration 19931 / 39200) loss: 2.194349
(Iteration 19941 / 39200) loss: 1.796922
(Iteration 19951 / 39200) loss: 1.391981
(Iteration 19961 / 39200) loss: 1.806639
(Iteration 19971 / 39200) loss: 1.576857
(Iteration 19981 / 39200) loss: 2.219621
(Iteration 19991 / 39200) loss: 2.324338
(Iteration 20001 / 39200) loss: 2.013517
(Iteration 20011 / 39200) loss: 1.908434
(Iteration 20021 / 39200) loss: 2.075138
(Iteration 20031 / 39200) loss: 1.823763
(Iteration 20041 / 39200) loss: 1.766100
(Iteration 20051 / 39200) loss: 1.706178
(Iteration 20061 / 39200) loss: 1.973965
(Iteration 20071 / 39200) loss: 1.729556
(Iteration 20081 / 39200) loss: 1.603811
(Iteration 20091 / 39200) loss: 1.743238
(Iteration 20101 / 39200) loss: 1.765671
(Iteration 20111 / 39200) loss: 1.729304
(Iteration 20121 / 39200) loss: 1.837242
(Iteration 20131 / 39200) loss: 1.666062
(Iteration 20141 / 39200) loss: 1.851159
(Iteration 20151 / 39200) loss: 1.498613
(Iteration 20161 / 39200) loss: 1.945117
(Iteration 20171 / 39200) loss: 1.718452
(Iteration 20181 / 39200) loss: 1.763103
(Iteration 20191 / 39200) loss: 2.096214
(Iteration 20201 / 39200) loss: 1.699809
(Iteration 20211 / 39200) loss: 2.017962
(Iteration 20221 / 39200) loss: 1.975241
(Iteration 20231 / 39200) loss: 1.667454
(Iteration 20241 / 39200) loss: 1.653890
(Iteration 20251 / 39200) loss: 1.968368
(Iteration 20261 / 39200) loss: 1.894168
(Iteration 20271 / 39200) loss: 1.774100
(Iteration 20281 / 39200) loss: 1.669384
(Iteration 20291 / 39200) loss: 2.052631
(Iteration 20301 / 39200) loss: 1.856719
(Iteration 20311 / 39200) loss: 2.050325
(Iteration 20321 / 39200) loss: 1.652340
(Iteration 20331 / 39200) loss: 1.935442
(Iteration 20341 / 39200) loss: 1.866665
```

```
(Iteration 20351 / 39200) loss: 1.377295
(Iteration 20361 / 39200) loss: 1.940444
(Iteration 20371 / 39200) loss: 1.646679
(Iteration 20381 / 39200) loss: 2.104504
(Iteration 20391 / 39200) loss: 1.952369
(Iteration 20401 / 39200) loss: 1.689929
(Iteration 20411 / 39200) loss: 1.398757
(Iteration 20421 / 39200) loss: 1.546298
(Iteration 20431 / 39200) loss: 1.750054
(Iteration 20441 / 39200) loss: 1.727606
(Iteration 20451 / 39200) loss: 1.273837
(Iteration 20461 / 39200) loss: 1.873422
(Iteration 20471 / 39200) loss: 1.790326
(Iteration 20481 / 39200) loss: 1.922326
(Iteration 20491 / 39200) loss: 1.696231
(Iteration 20501 / 39200) loss: 1.869633
(Iteration 20511 / 39200) loss: 1.603495
(Iteration 20521 / 39200) loss: 2.044249
(Iteration 20531 / 39200) loss: 1.231342
(Iteration 20541 / 39200) loss: 1.859438
(Iteration 20551 / 39200) loss: 1.662483
(Iteration 20561 / 39200) loss: 1.698327
(Iteration 20571 / 39200) loss: 1.448340
(Iteration 20581 / 39200) loss: 1.759477
(Iteration 20591 / 39200) loss: 1.824837
(Iteration 20601 / 39200) loss: 1.771792
(Iteration 20611 / 39200) loss: 1.796992
(Iteration 20621 / 39200) loss: 1.680620
(Iteration 20631 / 39200) loss: 1.755840
(Iteration 20641 / 39200) loss: 1.818089
(Iteration 20651 / 39200) loss: 1.627910
(Iteration 20661 / 39200) loss: 1.918644
(Iteration 20671 / 39200) loss: 1.630339
(Iteration 20681 / 39200) loss: 1.507144
(Iteration 20691 / 39200) loss: 1.695444
(Iteration 20701 / 39200) loss: 1.983131
(Iteration 20711 / 39200) loss: 2.195788
(Iteration 20721 / 39200) loss: 1.819280
(Iteration 20731 / 39200) loss: 1.754887
(Iteration 20741 / 39200) loss: 1.776492
(Iteration 20751 / 39200) loss: 1.612731
(Iteration 20761 / 39200) loss: 1.672749
(Iteration 20771 / 39200) loss: 1.749589
(Iteration 20781 / 39200) loss: 1.644643
(Iteration 20791 / 39200) loss: 1.700903
(Iteration 20801 / 39200) loss: 1.860550
(Iteration 20811 / 39200) loss: 1.488870
(Iteration 20821 / 39200) loss: 2.026102
```

```
(Iteration 20831 / 39200) loss: 1.604014
(Iteration 20841 / 39200) loss: 1.864279
(Iteration 20851 / 39200) loss: 1.579612
(Iteration 20861 / 39200) loss: 1.663120
(Iteration 20871 / 39200) loss: 1.654052
(Iteration 20881 / 39200) loss: 1.469524
(Iteration 20891 / 39200) loss: 1.930932
(Iteration 20901 / 39200) loss: 2.318644
(Iteration 20911 / 39200) loss: 1.662016
(Iteration 20921 / 39200) loss: 2.191447
(Iteration 20931 / 39200) loss: 1.777299
(Iteration 20941 / 39200) loss: 1.848104
(Iteration 20951 / 39200) loss: 1.716466
(Iteration 20961 / 39200) loss: 1.773800
(Iteration 20971 / 39200) loss: 1.728628
(Iteration 20981 / 39200) loss: 1.976031
(Iteration 20991 / 39200) loss: 1.514122
(Iteration 21001 / 39200) loss: 1.492335
(Iteration 21011 / 39200) loss: 1.784849
(Iteration 21021 / 39200) loss: 1.894171
(Iteration 21031 / 39200) loss: 1.588368
(Iteration 21041 / 39200) loss: 1.669725
(Iteration 21051 / 39200) loss: 1.463001
(Iteration 21061 / 39200) loss: 1.558619
(Iteration 21071 / 39200) loss: 1.716104
(Iteration 21081 / 39200) loss: 2.148219
(Iteration 21091 / 39200) loss: 1.787444
(Iteration 21101 / 39200) loss: 1.636636
(Iteration 21111 / 39200) loss: 1.246163
(Iteration 21121 / 39200) loss: 2.037289
(Iteration 21131 / 39200) loss: 1.471835
(Iteration 21141 / 39200) loss: 1.763415
(Iteration 21151 / 39200) loss: 1.553649
(Iteration 21161 / 39200) loss: 2.060695
(Iteration 21171 / 39200) loss: 1.961245
(Iteration 21181 / 39200) loss: 1.607819
(Iteration 21191 / 39200) loss: 2.065942
(Iteration 21201 / 39200) loss: 1.719885
(Iteration 21211 / 39200) loss: 2.182942
(Iteration 21221 / 39200) loss: 1.499580
(Iteration 21231 / 39200) loss: 1.616084
(Iteration 21241 / 39200) loss: 1.382006
(Iteration 21251 / 39200) loss: 1.418585
(Iteration 21261 / 39200) loss: 2.003599
(Iteration 21271 / 39200) loss: 1.768984
(Iteration 21281 / 39200) loss: 1.887326
(Iteration 21291 / 39200) loss: 2.033008
(Iteration 21301 / 39200) loss: 2.075438
```

```
(Iteration 21311 / 39200) loss: 1.773373
(Iteration 21321 / 39200) loss: 1.965901
(Iteration 21331 / 39200) loss: 1.382675
(Iteration 21341 / 39200) loss: 1.590374
(Iteration 21351 / 39200) loss: 1.406665
(Iteration 21361 / 39200) loss: 1.519046
(Iteration 21371 / 39200) loss: 1.634499
(Iteration 21381 / 39200) loss: 1.948571
(Iteration 21391 / 39200) loss: 1.758128
(Iteration 21401 / 39200) loss: 1.746620
(Iteration 21411 / 39200) loss: 1.944171
(Iteration 21421 / 39200) loss: 1.540265
(Iteration 21431 / 39200) loss: 1.377940
(Iteration 21441 / 39200) loss: 1.190386
(Iteration 21451 / 39200) loss: 1.717182
(Iteration 21461 / 39200) loss: 1.483015
(Iteration 21471 / 39200) loss: 1.994899
(Iteration 21481 / 39200) loss: 1.378804
(Iteration 21491 / 39200) loss: 1.322537
(Iteration 21501 / 39200) loss: 1.504577
(Iteration 21511 / 39200) loss: 1.511716
(Iteration 21521 / 39200) loss: 1.253673
(Iteration 21531 / 39200) loss: 1.779390
(Iteration 21541 / 39200) loss: 2.132285
(Iteration 21551 / 39200) loss: 1.506906
(Epoch 11 / 20) train acc: 0.598000; val_acc: 0.500000
(Iteration 21561 / 39200) loss: 1.787568
(Iteration 21571 / 39200) loss: 1.977989
(Iteration 21581 / 39200) loss: 1.558757
(Iteration 21591 / 39200) loss: 1.824417
(Iteration 21601 / 39200) loss: 1.732503
(Iteration 21611 / 39200) loss: 1.861068
(Iteration 21621 / 39200) loss: 1.878200
(Iteration 21631 / 39200) loss: 1.728186
(Iteration 21641 / 39200) loss: 1.623196
(Iteration 21651 / 39200) loss: 1.618523
(Iteration 21661 / 39200) loss: 1.359009
(Iteration 21671 / 39200) loss: 1.880314
(Iteration 21681 / 39200) loss: 1.723679
(Iteration 21691 / 39200) loss: 1.657964
(Iteration 21701 / 39200) loss: 1.505191
(Iteration 21711 / 39200) loss: 1.774071
(Iteration 21721 / 39200) loss: 1.613782
(Iteration 21731 / 39200) loss: 1.892243
(Iteration 21741 / 39200) loss: 1.649246
(Iteration 21751 / 39200) loss: 1.482289
(Iteration 21761 / 39200) loss: 2.085345
(Iteration 21771 / 39200) loss: 1.441196
```

```
(Iteration 21781 / 39200) loss: 1.251979
(Iteration 21791 / 39200) loss: 1.556559
(Iteration 21801 / 39200) loss: 1.843466
(Iteration 21811 / 39200) loss: 1.541913
(Iteration 21821 / 39200) loss: 1.288529
(Iteration 21831 / 39200) loss: 1.654144
(Iteration 21841 / 39200) loss: 2.143287
(Iteration 21851 / 39200) loss: 2.236757
(Iteration 21861 / 39200) loss: 2.114368
(Iteration 21871 / 39200) loss: 1.982191
(Iteration 21881 / 39200) loss: 2.052811
(Iteration 21891 / 39200) loss: 1.680389
(Iteration 21901 / 39200) loss: 1.594196
(Iteration 21911 / 39200) loss: 2.367944
(Iteration 21921 / 39200) loss: 1.882792
(Iteration 21931 / 39200) loss: 1.632849
(Iteration 21941 / 39200) loss: 1.519029
(Iteration 21951 / 39200) loss: 1.701769
(Iteration 21961 / 39200) loss: 1.769152
(Iteration 21971 / 39200) loss: 1.448523
(Iteration 21981 / 39200) loss: 1.471255
(Iteration 21991 / 39200) loss: 1.463480
(Iteration 22001 / 39200) loss: 2.177720
(Iteration 22011 / 39200) loss: 1.873284
(Iteration 22021 / 39200) loss: 1.965368
(Iteration 22031 / 39200) loss: 1.683762
(Iteration 22041 / 39200) loss: 1.778314
(Iteration 22051 / 39200) loss: 1.809327
(Iteration 22061 / 39200) loss: 2.064536
(Iteration 22071 / 39200) loss: 1.684533
(Iteration 22081 / 39200) loss: 1.956508
(Iteration 22091 / 39200) loss: 1.475963
(Iteration 22101 / 39200) loss: 1.664399
(Iteration 22111 / 39200) loss: 1.687826
(Iteration 22121 / 39200) loss: 1.811436
(Iteration 22131 / 39200) loss: 1.644635
(Iteration 22141 / 39200) loss: 1.410584
(Iteration 22151 / 39200) loss: 1.871689
(Iteration 22161 / 39200) loss: 2.152204
(Iteration 22171 / 39200) loss: 1.529866
(Iteration 22181 / 39200) loss: 1.844002
(Iteration 22191 / 39200) loss: 1.470211
(Iteration 22201 / 39200) loss: 1.608132
(Iteration 22211 / 39200) loss: 1.551701
(Iteration 22221 / 39200) loss: 1.658573
(Iteration 22231 / 39200) loss: 1.451704
(Iteration 22241 / 39200) loss: 1.702654
(Iteration 22251 / 39200) loss: 1.971409
```

```
(Iteration 22261 / 39200) loss: 1.945922
(Iteration 22271 / 39200) loss: 1.679230
(Iteration 22281 / 39200) loss: 1.792092
(Iteration 22291 / 39200) loss: 1.787280
(Iteration 22301 / 39200) loss: 1.622584
(Iteration 22311 / 39200) loss: 1.781265
(Iteration 22321 / 39200) loss: 1.395001
(Iteration 22331 / 39200) loss: 1.428057
(Iteration 22341 / 39200) loss: 1.403470
(Iteration 22351 / 39200) loss: 1.531352
(Iteration 22361 / 39200) loss: 1.820130
(Iteration 22371 / 39200) loss: 1.717875
(Iteration 22381 / 39200) loss: 1.669846
(Iteration 22391 / 39200) loss: 2.185660
(Iteration 22401 / 39200) loss: 1.888641
(Iteration 22411 / 39200) loss: 1.774258
(Iteration 22421 / 39200) loss: 1.862443
(Iteration 22431 / 39200) loss: 1.574533
(Iteration 22441 / 39200) loss: 1.958299
(Iteration 22451 / 39200) loss: 1.600654
(Iteration 22461 / 39200) loss: 2.038930
(Iteration 22471 / 39200) loss: 1.803846
(Iteration 22481 / 39200) loss: 2.011837
(Iteration 22491 / 39200) loss: 2.170306
(Iteration 22501 / 39200) loss: 1.573701
(Iteration 22511 / 39200) loss: 1.552708
(Iteration 22521 / 39200) loss: 1.565983
(Iteration 22531 / 39200) loss: 1.693776
(Iteration 22541 / 39200) loss: 1.572129
(Iteration 22551 / 39200) loss: 1.571137
(Iteration 22561 / 39200) loss: 1.854141
(Iteration 22571 / 39200) loss: 2.086566
(Iteration 22581 / 39200) loss: 2.139206
(Iteration 22591 / 39200) loss: 1.489330
(Iteration 22601 / 39200) loss: 1.661697
(Iteration 22611 / 39200) loss: 1.617274
(Iteration 22621 / 39200) loss: 1.577917
(Iteration 22631 / 39200) loss: 1.822391
(Iteration 22641 / 39200) loss: 1.645256
(Iteration 22651 / 39200) loss: 1.714668
(Iteration 22661 / 39200) loss: 1.463939
(Iteration 22671 / 39200) loss: 2.461798
(Iteration 22681 / 39200) loss: 1.823776
(Iteration 22691 / 39200) loss: 1.566555
(Iteration 22701 / 39200) loss: 2.062333
(Iteration 22711 / 39200) loss: 1.909656
(Iteration 22721 / 39200) loss: 1.664399
(Iteration 22731 / 39200) loss: 1.214975
```

```
(Iteration 22741 / 39200) loss: 1.361318
(Iteration 22751 / 39200) loss: 1.519579
(Iteration 22761 / 39200) loss: 1.861930
(Iteration 22771 / 39200) loss: 1.573375
(Iteration 22781 / 39200) loss: 1.957995
(Iteration 22791 / 39200) loss: 1.785952
(Iteration 22801 / 39200) loss: 1.420438
(Iteration 22811 / 39200) loss: 1.720021
(Iteration 22821 / 39200) loss: 1.424701
(Iteration 22831 / 39200) loss: 1.763531
(Iteration 22841 / 39200) loss: 2.240496
(Iteration 22851 / 39200) loss: 1.566961
(Iteration 22861 / 39200) loss: 1.669840
(Iteration 22871 / 39200) loss: 1.600221
(Iteration 22881 / 39200) loss: 1.674878
(Iteration 22891 / 39200) loss: 1.883230
(Iteration 22901 / 39200) loss: 1.745309
(Iteration 22911 / 39200) loss: 1.594591
(Iteration 22921 / 39200) loss: 1.960206
(Iteration 22931 / 39200) loss: 1.548311
(Iteration 22941 / 39200) loss: 1.467724
(Iteration 22951 / 39200) loss: 1.717605
(Iteration 22961 / 39200) loss: 1.380540
(Iteration 22971 / 39200) loss: 1.525124
(Iteration 22981 / 39200) loss: 1.677308
(Iteration 22991 / 39200) loss: 1.599288
(Iteration 23001 / 39200) loss: 1.806796
(Iteration 23011 / 39200) loss: 1.683978
(Iteration 23021 / 39200) loss: 1.817921
(Iteration 23031 / 39200) loss: 2.093964
(Iteration 23041 / 39200) loss: 1.925555
(Iteration 23051 / 39200) loss: 1.558700
(Iteration 23061 / 39200) loss: 1.525011
(Iteration 23071 / 39200) loss: 1.990479
(Iteration 23081 / 39200) loss: 1.526528
(Iteration 23091 / 39200) loss: 1.496007
(Iteration 23101 / 39200) loss: 1.595761
(Iteration 23111 / 39200) loss: 1.898603
(Iteration 23121 / 39200) loss: 1.819046
(Iteration 23131 / 39200) loss: 1.612401
(Iteration 23141 / 39200) loss: 1.615155
(Iteration 23151 / 39200) loss: 1.887821
(Iteration 23161 / 39200) loss: 1.505237
(Iteration 23171 / 39200) loss: 2.023177
(Iteration 23181 / 39200) loss: 1.664518
(Iteration 23191 / 39200) loss: 2.009972
(Iteration 23201 / 39200) loss: 1.645037
(Iteration 23211 / 39200) loss: 1.603524
```

```
(Iteration 23221 / 39200) loss: 1.971531
(Iteration 23231 / 39200) loss: 1.636215
(Iteration 23241 / 39200) loss: 1.863896
(Iteration 23251 / 39200) loss: 1.661657
(Iteration 23261 / 39200) loss: 1.558146
(Iteration 23271 / 39200) loss: 1.630825
(Iteration 23281 / 39200) loss: 1.789476
(Iteration 23291 / 39200) loss: 1.777429
(Iteration 23301 / 39200) loss: 1.583730
(Iteration 23311 / 39200) loss: 1.347641
(Iteration 23321 / 39200) loss: 1.513524
(Iteration 23331 / 39200) loss: 1.389939
(Iteration 23341 / 39200) loss: 1.719987
(Iteration 23351 / 39200) loss: 1.291414
(Iteration 23361 / 39200) loss: 1.524289
(Iteration 23371 / 39200) loss: 1.733329
(Iteration 23381 / 39200) loss: 1.908351
(Iteration 23391 / 39200) loss: 1.411837
(Iteration 23401 / 39200) loss: 2.274147
(Iteration 23411 / 39200) loss: 1.930063
(Iteration 23421 / 39200) loss: 1.720041
(Iteration 23431 / 39200) loss: 1.710037
(Iteration 23441 / 39200) loss: 1.406432
(Iteration 23451 / 39200) loss: 1.870154
(Iteration 23461 / 39200) loss: 1.718124
(Iteration 23471 / 39200) loss: 1.452885
(Iteration 23481 / 39200) loss: 2.151435
(Iteration 23491 / 39200) loss: 1.938448
(Iteration 23501 / 39200) loss: 1.781877
(Iteration 23511 / 39200) loss: 1.603071
(Epoch 12 / 20) train acc: 0.619000; val_acc: 0.474000
(Iteration 23521 / 39200) loss: 1.584157
(Iteration 23531 / 39200) loss: 1.714105
(Iteration 23541 / 39200) loss: 1.590269
(Iteration 23551 / 39200) loss: 1.590042
(Iteration 23561 / 39200) loss: 1.562870
(Iteration 23571 / 39200) loss: 1.746239
(Iteration 23581 / 39200) loss: 1.848681
(Iteration 23591 / 39200) loss: 1.434510
(Iteration 23601 / 39200) loss: 1.802430
(Iteration 23611 / 39200) loss: 1.739301
(Iteration 23621 / 39200) loss: 1.851673
(Iteration 23631 / 39200) loss: 1.345568
(Iteration 23641 / 39200) loss: 1.515355
(Iteration 23651 / 39200) loss: 1.623053
(Iteration 23661 / 39200) loss: 1.571146
(Iteration 23671 / 39200) loss: 1.404053
(Iteration 23681 / 39200) loss: 1.842370
```

```
(Iteration 23691 / 39200) loss: 1.837087
(Iteration 23701 / 39200) loss: 1.825987
(Iteration 23711 / 39200) loss: 1.536539
(Iteration 23721 / 39200) loss: 1.590500
(Iteration 23731 / 39200) loss: 1.506886
(Iteration 23741 / 39200) loss: 1.953749
(Iteration 23751 / 39200) loss: 2.521592
(Iteration 23761 / 39200) loss: 1.961081
(Iteration 23771 / 39200) loss: 1.414596
(Iteration 23781 / 39200) loss: 1.573505
(Iteration 23791 / 39200) loss: 1.747731
(Iteration 23801 / 39200) loss: 1.649177
(Iteration 23811 / 39200) loss: 1.734749
(Iteration 23821 / 39200) loss: 1.595169
(Iteration 23831 / 39200) loss: 1.572287
(Iteration 23841 / 39200) loss: 1.735977
(Iteration 23851 / 39200) loss: 1.313418
(Iteration 23861 / 39200) loss: 2.157019
(Iteration 23871 / 39200) loss: 1.497642
(Iteration 23881 / 39200) loss: 2.050449
(Iteration 23891 / 39200) loss: 1.524599
(Iteration 23901 / 39200) loss: 1.548933
(Iteration 23911 / 39200) loss: 1.813369
(Iteration 23921 / 39200) loss: 1.377679
(Iteration 23931 / 39200) loss: 1.970122
(Iteration 23941 / 39200) loss: 1.751349
(Iteration 23951 / 39200) loss: 1.848537
(Iteration 23961 / 39200) loss: 1.486542
(Iteration 23971 / 39200) loss: 1.482910
(Iteration 23981 / 39200) loss: 1.666918
(Iteration 23991 / 39200) loss: 1.647650
(Iteration 24001 / 39200) loss: 2.385004
(Iteration 24011 / 39200) loss: 1.932502
(Iteration 24021 / 39200) loss: 1.667366
(Iteration 24031 / 39200) loss: 1.634743
(Iteration 24041 / 39200) loss: 1.431772
(Iteration 24051 / 39200) loss: 1.382966
(Iteration 24061 / 39200) loss: 2.013693
(Iteration 24071 / 39200) loss: 1.866715
(Iteration 24081 / 39200) loss: 1.683324
(Iteration 24091 / 39200) loss: 1.751556
(Iteration 24101 / 39200) loss: 1.779171
(Iteration 24111 / 39200) loss: 1.672393
(Iteration 24121 / 39200) loss: 1.586319
(Iteration 24131 / 39200) loss: 1.376054
(Iteration 24141 / 39200) loss: 1.445207
(Iteration 24151 / 39200) loss: 1.454880
(Iteration 24161 / 39200) loss: 1.487605
```

```
(Iteration 24171 / 39200) loss: 1.677617
(Iteration 24181 / 39200) loss: 1.752799
(Iteration 24191 / 39200) loss: 1.633824
(Iteration 24201 / 39200) loss: 1.448694
(Iteration 24211 / 39200) loss: 1.897271
(Iteration 24221 / 39200) loss: 1.877760
(Iteration 24231 / 39200) loss: 1.255182
(Iteration 24241 / 39200) loss: 1.247073
(Iteration 24251 / 39200) loss: 1.678124
(Iteration 24261 / 39200) loss: 1.502356
(Iteration 24271 / 39200) loss: 1.731856
(Iteration 24281 / 39200) loss: 1.667365
(Iteration 24291 / 39200) loss: 2.070353
(Iteration 24301 / 39200) loss: 1.692760
(Iteration 24311 / 39200) loss: 1.905550
(Iteration 24321 / 39200) loss: 1.439068
(Iteration 24331 / 39200) loss: 2.115578
(Iteration 24341 / 39200) loss: 1.877944
(Iteration 24351 / 39200) loss: 1.640708
(Iteration 24361 / 39200) loss: 1.764324
(Iteration 24371 / 39200) loss: 1.516309
(Iteration 24381 / 39200) loss: 1.431165
(Iteration 24391 / 39200) loss: 1.625909
(Iteration 24401 / 39200) loss: 1.217526
(Iteration 24411 / 39200) loss: 1.583413
(Iteration 24421 / 39200) loss: 1.678846
(Iteration 24431 / 39200) loss: 2.088230
(Iteration 24441 / 39200) loss: 1.795594
(Iteration 24451 / 39200) loss: 1.381466
(Iteration 24461 / 39200) loss: 1.688022
(Iteration 24471 / 39200) loss: 1.960321
(Iteration 24481 / 39200) loss: 1.653240
(Iteration 24491 / 39200) loss: 1.610221
(Iteration 24501 / 39200) loss: 1.998355
(Iteration 24511 / 39200) loss: 1.494803
(Iteration 24521 / 39200) loss: 1.887694
(Iteration 24531 / 39200) loss: 1.435486
(Iteration 24541 / 39200) loss: 1.708138
(Iteration 24551 / 39200) loss: 1.370772
(Iteration 24561 / 39200) loss: 1.976521
(Iteration 24571 / 39200) loss: 1.489614
(Iteration 24581 / 39200) loss: 1.810152
(Iteration 24591 / 39200) loss: 1.377043
(Iteration 24601 / 39200) loss: 1.919315
(Iteration 24611 / 39200) loss: 1.461308
(Iteration 24621 / 39200) loss: 1.505303
(Iteration 24631 / 39200) loss: 2.014594
(Iteration 24641 / 39200) loss: 1.749590
```

```
(Iteration 24651 / 39200) loss: 1.678100
(Iteration 24661 / 39200) loss: 1.860118
(Iteration 24671 / 39200) loss: 1.334107
(Iteration 24681 / 39200) loss: 1.265804
(Iteration 24691 / 39200) loss: 1.424475
(Iteration 24701 / 39200) loss: 1.471826
(Iteration 24711 / 39200) loss: 1.648217
(Iteration 24721 / 39200) loss: 1.612743
(Iteration 24731 / 39200) loss: 1.384450
(Iteration 24741 / 39200) loss: 1.298383
(Iteration 24751 / 39200) loss: 1.446314
(Iteration 24761 / 39200) loss: 1.714351
(Iteration 24771 / 39200) loss: 1.493348
(Iteration 24781 / 39200) loss: 1.406761
(Iteration 24791 / 39200) loss: 1.728762
(Iteration 24801 / 39200) loss: 1.319217
(Iteration 24811 / 39200) loss: 1.795522
(Iteration 24821 / 39200) loss: 1.777331
(Iteration 24831 / 39200) loss: 1.564355
(Iteration 24841 / 39200) loss: 1.592798
(Iteration 24851 / 39200) loss: 1.599392
(Iteration 24861 / 39200) loss: 1.789638
(Iteration 24871 / 39200) loss: 1.623860
(Iteration 24881 / 39200) loss: 1.862638
(Iteration 24891 / 39200) loss: 1.439669
(Iteration 24901 / 39200) loss: 1.806600
(Iteration 24911 / 39200) loss: 1.502322
(Iteration 24921 / 39200) loss: 1.695844
(Iteration 24931 / 39200) loss: 1.397288
(Iteration 24941 / 39200) loss: 1.628425
(Iteration 24951 / 39200) loss: 1.751955
(Iteration 24961 / 39200) loss: 1.595109
(Iteration 24971 / 39200) loss: 1.666536
(Iteration 24981 / 39200) loss: 1.687762
(Iteration 24991 / 39200) loss: 1.545560
(Iteration 25001 / 39200) loss: 1.716748
(Iteration 25011 / 39200) loss: 1.654165
(Iteration 25021 / 39200) loss: 1.735823
(Iteration 25031 / 39200) loss: 1.104280
(Iteration 25041 / 39200) loss: 1.789399
(Iteration 25051 / 39200) loss: 1.498516
(Iteration 25061 / 39200) loss: 1.937765
(Iteration 25071 / 39200) loss: 1.664531
(Iteration 25081 / 39200) loss: 1.504784
(Iteration 25091 / 39200) loss: 1.642666
(Iteration 25101 / 39200) loss: 1.741622
(Iteration 25111 / 39200) loss: 1.723672
(Iteration 25121 / 39200) loss: 2.098408
```

```
(Iteration 25131 / 39200) loss: 1.994877
(Iteration 25141 / 39200) loss: 1.535454
(Iteration 25151 / 39200) loss: 1.636422
(Iteration 25161 / 39200) loss: 1.661067
(Iteration 25171 / 39200) loss: 1.533909
(Iteration 25181 / 39200) loss: 1.421441
(Iteration 25191 / 39200) loss: 1.523527
(Iteration 25201 / 39200) loss: 1.613984
(Iteration 25211 / 39200) loss: 1.570546
(Iteration 25221 / 39200) loss: 1.432405
(Iteration 25231 / 39200) loss: 1.960249
(Iteration 25241 / 39200) loss: 1.713295
(Iteration 25251 / 39200) loss: 1.466925
(Iteration 25261 / 39200) loss: 1.296749
(Iteration 25271 / 39200) loss: 1.371967
(Iteration 25281 / 39200) loss: 1.672752
(Iteration 25291 / 39200) loss: 1.968429
(Iteration 25301 / 39200) loss: 1.312413
(Iteration 25311 / 39200) loss: 1.487405
(Iteration 25321 / 39200) loss: 1.613332
(Iteration 25331 / 39200) loss: 1.542504
(Iteration 25341 / 39200) loss: 1.743573
(Iteration 25351 / 39200) loss: 1.373612
(Iteration 25361 / 39200) loss: 1.867606
(Iteration 25371 / 39200) loss: 1.304841
(Iteration 25381 / 39200) loss: 1.510226
(Iteration 25391 / 39200) loss: 1.600079
(Iteration 25401 / 39200) loss: 1.691282
(Iteration 25411 / 39200) loss: 1.783686
(Iteration 25421 / 39200) loss: 1.465224
(Iteration 25431 / 39200) loss: 1.775734
(Iteration 25441 / 39200) loss: 1.629180
(Iteration 25451 / 39200) loss: 1.272384
(Iteration 25461 / 39200) loss: 1.498718
(Iteration 25471 / 39200) loss: 1.337701
(Epoch 13 / 20) train acc: 0.605000; val_acc: 0.484000
(Iteration 25481 / 39200) loss: 1.772828
(Iteration 25491 / 39200) loss: 1.401769
(Iteration 25501 / 39200) loss: 1.418815
(Iteration 25511 / 39200) loss: 1.754280
(Iteration 25521 / 39200) loss: 1.478274
(Iteration 25531 / 39200) loss: 1.559461
(Iteration 25541 / 39200) loss: 1.681265
(Iteration 25551 / 39200) loss: 1.122819
(Iteration 25561 / 39200) loss: 1.849651
(Iteration 25571 / 39200) loss: 2.101953
(Iteration 25581 / 39200) loss: 1.718605
(Iteration 25591 / 39200) loss: 1.495057
```

```
(Iteration 25601 / 39200) loss: 1.921073
(Iteration 25611 / 39200) loss: 1.423916
(Iteration 25621 / 39200) loss: 2.264885
(Iteration 25631 / 39200) loss: 1.518850
(Iteration 25641 / 39200) loss: 1.671009
(Iteration 25651 / 39200) loss: 1.805992
(Iteration 25661 / 39200) loss: 1.348613
(Iteration 25671 / 39200) loss: 1.909214
(Iteration 25681 / 39200) loss: 1.509331
(Iteration 25691 / 39200) loss: 1.607944
(Iteration 25701 / 39200) loss: 1.960550
(Iteration 25711 / 39200) loss: 1.339117
(Iteration 25721 / 39200) loss: 1.820871
(Iteration 25731 / 39200) loss: 1.933070
(Iteration 25741 / 39200) loss: 1.407622
(Iteration 25751 / 39200) loss: 1.772901
(Iteration 25761 / 39200) loss: 1.744838
(Iteration 25771 / 39200) loss: 1.674794
(Iteration 25781 / 39200) loss: 1.582115
(Iteration 25791 / 39200) loss: 1.440226
(Iteration 25801 / 39200) loss: 1.478965
(Iteration 25811 / 39200) loss: 1.909705
(Iteration 25821 / 39200) loss: 1.294328
(Iteration 25831 / 39200) loss: 1.528302
(Iteration 25841 / 39200) loss: 1.618199
(Iteration 25851 / 39200) loss: 1.508340
(Iteration 25861 / 39200) loss: 1.188347
(Iteration 25871 / 39200) loss: 1.201803
(Iteration 25881 / 39200) loss: 1.379147
(Iteration 25891 / 39200) loss: 1.743685
(Iteration 25901 / 39200) loss: 1.431547
(Iteration 25911 / 39200) loss: 1.404490
(Iteration 25921 / 39200) loss: 1.907653
(Iteration 25931 / 39200) loss: 1.527554
(Iteration 25941 / 39200) loss: 1.290411
(Iteration 25951 / 39200) loss: 1.601625
(Iteration 25961 / 39200) loss: 1.660276
(Iteration 25971 / 39200) loss: 1.683731
(Iteration 25981 / 39200) loss: 1.696459
(Iteration 25991 / 39200) loss: 1.486769
(Iteration 26001 / 39200) loss: 1.599121
(Iteration 26011 / 39200) loss: 1.655681
(Iteration 26021 / 39200) loss: 2.049150
(Iteration 26031 / 39200) loss: 1.660399
(Iteration 26041 / 39200) loss: 1.796640
(Iteration 26051 / 39200) loss: 1.524236
(Iteration 26061 / 39200) loss: 1.966543
(Iteration 26071 / 39200) loss: 1.687755
```

```
(Iteration 26081 / 39200) loss: 1.612987
(Iteration 26091 / 39200) loss: 1.819368
(Iteration 26101 / 39200) loss: 1.614337
(Iteration 26111 / 39200) loss: 1.567879
(Iteration 26121 / 39200) loss: 1.630444
(Iteration 26131 / 39200) loss: 1.516893
(Iteration 26141 / 39200) loss: 1.540906
(Iteration 26151 / 39200) loss: 1.389002
(Iteration 26161 / 39200) loss: 1.965768
(Iteration 26171 / 39200) loss: 1.854735
(Iteration 26181 / 39200) loss: 2.051564
(Iteration 26191 / 39200) loss: 1.703919
(Iteration 26201 / 39200) loss: 1.391525
(Iteration 26211 / 39200) loss: 1.089175
(Iteration 26221 / 39200) loss: 1.491438
(Iteration 26231 / 39200) loss: 1.724003
(Iteration 26241 / 39200) loss: 1.586652
(Iteration 26251 / 39200) loss: 1.624436
(Iteration 26261 / 39200) loss: 1.563424
(Iteration 26271 / 39200) loss: 1.616769
(Iteration 26281 / 39200) loss: 1.524909
(Iteration 26291 / 39200) loss: 1.337586
(Iteration 26301 / 39200) loss: 1.747976
(Iteration 26311 / 39200) loss: 1.662928
(Iteration 26321 / 39200) loss: 1.648018
(Iteration 26331 / 39200) loss: 1.347863
(Iteration 26341 / 39200) loss: 1.729597
(Iteration 26351 / 39200) loss: 1.800815
(Iteration 26361 / 39200) loss: 1.780131
(Iteration 26371 / 39200) loss: 1.069992
(Iteration 26381 / 39200) loss: 1.750399
(Iteration 26391 / 39200) loss: 1.614124
(Iteration 26401 / 39200) loss: 1.726925
(Iteration 26411 / 39200) loss: 1.451896
(Iteration 26421 / 39200) loss: 1.386057
(Iteration 26431 / 39200) loss: 1.548053
(Iteration 26441 / 39200) loss: 1.814014
(Iteration 26451 / 39200) loss: 1.846989
(Iteration 26461 / 39200) loss: 1.965217
(Iteration 26471 / 39200) loss: 1.520579
(Iteration 26481 / 39200) loss: 1.680069
(Iteration 26491 / 39200) loss: 1.269807
(Iteration 26501 / 39200) loss: 1.639710
(Iteration 26511 / 39200) loss: 1.544447
(Iteration 26521 / 39200) loss: 1.059853
(Iteration 26531 / 39200) loss: 1.716677
(Iteration 26541 / 39200) loss: 1.109100
(Iteration 26551 / 39200) loss: 1.711250
```

```
(Iteration 26561 / 39200) loss: 1.416119
(Iteration 26571 / 39200) loss: 1.673088
(Iteration 26581 / 39200) loss: 1.098435
(Iteration 26591 / 39200) loss: 1.667771
(Iteration 26601 / 39200) loss: 1.397834
(Iteration 26611 / 39200) loss: 1.413969
(Iteration 26621 / 39200) loss: 1.370656
(Iteration 26631 / 39200) loss: 1.536125
(Iteration 26641 / 39200) loss: 1.513334
(Iteration 26651 / 39200) loss: 1.626757
(Iteration 26661 / 39200) loss: 1.508069
(Iteration 26671 / 39200) loss: 1.441217
(Iteration 26681 / 39200) loss: 1.316748
(Iteration 26691 / 39200) loss: 1.358322
(Iteration 26701 / 39200) loss: 1.622431
(Iteration 26711 / 39200) loss: 1.847679
(Iteration 26721 / 39200) loss: 1.432926
(Iteration 26731 / 39200) loss: 1.193965
(Iteration 26741 / 39200) loss: 1.202034
(Iteration 26751 / 39200) loss: 1.655202
(Iteration 26761 / 39200) loss: 1.867509
(Iteration 26771 / 39200) loss: 1.517482
(Iteration 26781 / 39200) loss: 1.384396
(Iteration 26791 / 39200) loss: 1.876914
(Iteration 26801 / 39200) loss: 1.388196
(Iteration 26811 / 39200) loss: 1.513421
(Iteration 26821 / 39200) loss: 1.430016
(Iteration 26831 / 39200) loss: 1.136354
(Iteration 26841 / 39200) loss: 1.345973
(Iteration 26851 / 39200) loss: 1.785402
(Iteration 26861 / 39200) loss: 1.409730
(Iteration 26871 / 39200) loss: 1.273450
(Iteration 26881 / 39200) loss: 1.511317
(Iteration 26891 / 39200) loss: 2.041473
(Iteration 26901 / 39200) loss: 1.662428
(Iteration 26911 / 39200) loss: 1.415793
(Iteration 26921 / 39200) loss: 1.754536
(Iteration 26931 / 39200) loss: 1.996997
(Iteration 26941 / 39200) loss: 2.131720
(Iteration 26951 / 39200) loss: 1.793580
(Iteration 26961 / 39200) loss: 1.342301
(Iteration 26971 / 39200) loss: 1.367579
(Iteration 26981 / 39200) loss: 1.866187
(Iteration 26991 / 39200) loss: 1.310968
(Iteration 27001 / 39200) loss: 1.410361
(Iteration 27011 / 39200) loss: 1.505073
(Iteration 27021 / 39200) loss: 1.590594
(Iteration 27031 / 39200) loss: 1.692217
```

```
(Iteration 27041 / 39200) loss: 1.282116
(Iteration 27051 / 39200) loss: 1.429659
(Iteration 27061 / 39200) loss: 1.518740
(Iteration 27071 / 39200) loss: 1.477694
(Iteration 27081 / 39200) loss: 1.716989
(Iteration 27091 / 39200) loss: 1.872642
(Iteration 27101 / 39200) loss: 1.446971
(Iteration 27111 / 39200) loss: 1.704719
(Iteration 27121 / 39200) loss: 1.215987
(Iteration 27131 / 39200) loss: 1.703142
(Iteration 27141 / 39200) loss: 1.921251
(Iteration 27151 / 39200) loss: 1.676692
(Iteration 27161 / 39200) loss: 1.944975
(Iteration 27171 / 39200) loss: 1.821977
(Iteration 27181 / 39200) loss: 1.734953
(Iteration 27191 / 39200) loss: 1.581726
(Iteration 27201 / 39200) loss: 1.632176
(Iteration 27211 / 39200) loss: 1.579548
(Iteration 27221 / 39200) loss: 1.399870
(Iteration 27231 / 39200) loss: 1.553380
(Iteration 27241 / 39200) loss: 1.611332
(Iteration 27251 / 39200) loss: 1.566010
(Iteration 27261 / 39200) loss: 1.384849
(Iteration 27271 / 39200) loss: 1.799292
(Iteration 27281 / 39200) loss: 1.485374
(Iteration 27291 / 39200) loss: 1.664051
(Iteration 27301 / 39200) loss: 1.525046
(Iteration 27311 / 39200) loss: 1.680587
(Iteration 27321 / 39200) loss: 1.562526
(Iteration 27331 / 39200) loss: 1.641192
(Iteration 27341 / 39200) loss: 2.153057
(Iteration 27351 / 39200) loss: 1.359649
(Iteration 27361 / 39200) loss: 1.641940
(Iteration 27371 / 39200) loss: 1.651557
(Iteration 27381 / 39200) loss: 1.842799
(Iteration 27391 / 39200) loss: 1.710024
(Iteration 27401 / 39200) loss: 1.313736
(Iteration 27411 / 39200) loss: 1.705799
(Iteration 27421 / 39200) loss: 1.697578
(Iteration 27431 / 39200) loss: 1.170337
(Epoch 14 / 20) train acc: 0.618000; val_acc: 0.492000
(Iteration 27441 / 39200) loss: 1.554590
(Iteration 27451 / 39200) loss: 1.415272
(Iteration 27461 / 39200) loss: 1.250144
(Iteration 27471 / 39200) loss: 2.138121
(Iteration 27481 / 39200) loss: 1.400203
(Iteration 27491 / 39200) loss: 1.430208
(Iteration 27501 / 39200) loss: 1.532230
```

```
(Iteration 27511 / 39200) loss: 1.597937
(Iteration 27521 / 39200) loss: 1.319996
(Iteration 27531 / 39200) loss: 1.389293
(Iteration 27541 / 39200) loss: 1.431130
(Iteration 27551 / 39200) loss: 1.650804
(Iteration 27561 / 39200) loss: 1.278751
(Iteration 27571 / 39200) loss: 1.258592
(Iteration 27581 / 39200) loss: 1.492141
(Iteration 27591 / 39200) loss: 1.637451
(Iteration 27601 / 39200) loss: 1.929671
(Iteration 27611 / 39200) loss: 1.283539
(Iteration 27621 / 39200) loss: 1.578169
(Iteration 27631 / 39200) loss: 1.528764
(Iteration 27641 / 39200) loss: 1.642251
(Iteration 27651 / 39200) loss: 1.263627
(Iteration 27661 / 39200) loss: 1.808061
(Iteration 27671 / 39200) loss: 1.260602
(Iteration 27681 / 39200) loss: 1.322965
(Iteration 27691 / 39200) loss: 1.352172
(Iteration 27701 / 39200) loss: 1.451791
(Iteration 27711 / 39200) loss: 1.687709
(Iteration 27721 / 39200) loss: 1.302287
(Iteration 27731 / 39200) loss: 1.368990
(Iteration 27741 / 39200) loss: 1.300904
(Iteration 27751 / 39200) loss: 2.048751
(Iteration 27761 / 39200) loss: 1.219478
(Iteration 27771 / 39200) loss: 1.761752
(Iteration 27781 / 39200) loss: 1.489412
(Iteration 27791 / 39200) loss: 1.960387
(Iteration 27801 / 39200) loss: 1.793433
(Iteration 27811 / 39200) loss: 1.303946
(Iteration 27821 / 39200) loss: 1.418016
(Iteration 27831 / 39200) loss: 1.517795
(Iteration 27841 / 39200) loss: 1.983488
(Iteration 27851 / 39200) loss: 1.710656
(Iteration 27861 / 39200) loss: 1.235267
(Iteration 27871 / 39200) loss: 1.546928
(Iteration 27881 / 39200) loss: 2.183690
(Iteration 27891 / 39200) loss: 1.652975
(Iteration 27901 / 39200) loss: 1.556188
(Iteration 27911 / 39200) loss: 1.347123
(Iteration 27921 / 39200) loss: 1.631614
(Iteration 27931 / 39200) loss: 1.299682
(Iteration 27941 / 39200) loss: 1.521245
(Iteration 27951 / 39200) loss: 1.857923
(Iteration 27961 / 39200) loss: 1.278274
(Iteration 27971 / 39200) loss: 1.306354
(Iteration 27981 / 39200) loss: 1.751673
```

```
(Iteration 27991 / 39200) loss: 1.470007
(Iteration 28001 / 39200) loss: 1.641592
(Iteration 28011 / 39200) loss: 1.578780
(Iteration 28021 / 39200) loss: 1.518369
(Iteration 28031 / 39200) loss: 1.457198
(Iteration 28041 / 39200) loss: 1.578612
(Iteration 28051 / 39200) loss: 1.831536
(Iteration 28061 / 39200) loss: 1.453571
(Iteration 28071 / 39200) loss: 1.981238
(Iteration 28081 / 39200) loss: 1.271489
(Iteration 28091 / 39200) loss: 1.644474
(Iteration 28101 / 39200) loss: 1.223135
(Iteration 28111 / 39200) loss: 1.631616
(Iteration 28121 / 39200) loss: 1.647287
(Iteration 28131 / 39200) loss: 1.221514
(Iteration 28141 / 39200) loss: 1.826253
(Iteration 28151 / 39200) loss: 1.623224
(Iteration 28161 / 39200) loss: 1.249257
(Iteration 28171 / 39200) loss: 1.358378
(Iteration 28181 / 39200) loss: 1.714946
(Iteration 28191 / 39200) loss: 1.687362
(Iteration 28201 / 39200) loss: 1.557488
(Iteration 28211 / 39200) loss: 1.691570
(Iteration 28221 / 39200) loss: 1.281235
(Iteration 28231 / 39200) loss: 1.902091
(Iteration 28241 / 39200) loss: 1.525080
(Iteration 28251 / 39200) loss: 1.947200
(Iteration 28261 / 39200) loss: 1.365706
(Iteration 28271 / 39200) loss: 1.163847
(Iteration 28281 / 39200) loss: 1.829974
(Iteration 28291 / 39200) loss: 1.276661
(Iteration 28301 / 39200) loss: 1.415195
(Iteration 28311 / 39200) loss: 1.470682
(Iteration 28321 / 39200) loss: 1.778639
(Iteration 28331 / 39200) loss: 1.609908
(Iteration 28341 / 39200) loss: 1.456669
(Iteration 28351 / 39200) loss: 1.378456
(Iteration 28361 / 39200) loss: 1.805733
(Iteration 28371 / 39200) loss: 1.377264
(Iteration 28381 / 39200) loss: 1.456955
(Iteration 28391 / 39200) loss: 1.577711
(Iteration 28401 / 39200) loss: 1.749738
(Iteration 28411 / 39200) loss: 1.791134
(Iteration 28421 / 39200) loss: 1.648888
(Iteration 28431 / 39200) loss: 1.716729
(Iteration 28441 / 39200) loss: 1.703338
(Iteration 28451 / 39200) loss: 1.499026
(Iteration 28461 / 39200) loss: 1.849890
```

```
(Iteration 28471 / 39200) loss: 1.739400
(Iteration 28481 / 39200) loss: 1.289171
(Iteration 28491 / 39200) loss: 1.688551
(Iteration 28501 / 39200) loss: 1.835101
(Iteration 28511 / 39200) loss: 1.436645
(Iteration 28521 / 39200) loss: 1.561106
(Iteration 28531 / 39200) loss: 1.299394
(Iteration 28541 / 39200) loss: 1.416549
(Iteration 28551 / 39200) loss: 1.489919
(Iteration 28561 / 39200) loss: 1.595759
(Iteration 28571 / 39200) loss: 2.124619
(Iteration 28581 / 39200) loss: 1.005410
(Iteration 28591 / 39200) loss: 1.924741
(Iteration 28601 / 39200) loss: 1.410823
(Iteration 28611 / 39200) loss: 1.345467
(Iteration 28621 / 39200) loss: 1.425599
(Iteration 28631 / 39200) loss: 1.696429
(Iteration 28641 / 39200) loss: 2.069623
(Iteration 28651 / 39200) loss: 1.796312
(Iteration 28661 / 39200) loss: 1.666376
(Iteration 28671 / 39200) loss: 1.438053
(Iteration 28681 / 39200) loss: 1.339193
(Iteration 28691 / 39200) loss: 1.413658
(Iteration 28701 / 39200) loss: 1.854854
(Iteration 28711 / 39200) loss: 1.263770
(Iteration 28721 / 39200) loss: 1.555923
(Iteration 28731 / 39200) loss: 1.455798
(Iteration 28741 / 39200) loss: 1.546186
(Iteration 28751 / 39200) loss: 1.324310
(Iteration 28761 / 39200) loss: 1.240951
(Iteration 28771 / 39200) loss: 1.652438
(Iteration 28781 / 39200) loss: 1.641888
(Iteration 28791 / 39200) loss: 1.916154
(Iteration 28801 / 39200) loss: 1.472634
(Iteration 28811 / 39200) loss: 1.454035
(Iteration 28821 / 39200) loss: 1.323865
(Iteration 28831 / 39200) loss: 1.432290
(Iteration 28841 / 39200) loss: 1.850440
(Iteration 28851 / 39200) loss: 1.528884
(Iteration 28861 / 39200) loss: 2.131162
(Iteration 28871 / 39200) loss: 1.652010
(Iteration 28881 / 39200) loss: 1.335757
(Iteration 28891 / 39200) loss: 1.556580
(Iteration 28901 / 39200) loss: 1.594924
(Iteration 28911 / 39200) loss: 1.993133
(Iteration 28921 / 39200) loss: 1.353585
(Iteration 28931 / 39200) loss: 0.933971
(Iteration 28941 / 39200) loss: 1.608538
```

```
(Iteration 28951 / 39200) loss: 1.477508
(Iteration 28961 / 39200) loss: 1.608537
(Iteration 28971 / 39200) loss: 1.576947
(Iteration 28981 / 39200) loss: 1.194707
(Iteration 28991 / 39200) loss: 1.480908
(Iteration 29001 / 39200) loss: 1.612931
(Iteration 29011 / 39200) loss: 1.115704
(Iteration 29021 / 39200) loss: 1.819318
(Iteration 29031 / 39200) loss: 1.480604
(Iteration 29041 / 39200) loss: 1.686916
(Iteration 29051 / 39200) loss: 1.324293
(Iteration 29061 / 39200) loss: 1.275609
(Iteration 29071 / 39200) loss: 1.379254
(Iteration 29081 / 39200) loss: 1.918811
(Iteration 29091 / 39200) loss: 1.588546
(Iteration 29101 / 39200) loss: 1.628623
(Iteration 29111 / 39200) loss: 1.440220
(Iteration 29121 / 39200) loss: 1.430676
(Iteration 29131 / 39200) loss: 1.510679
(Iteration 29141 / 39200) loss: 1.565249
(Iteration 29151 / 39200) loss: 1.894122
(Iteration 29161 / 39200) loss: 1.153987
(Iteration 29171 / 39200) loss: 1.371185
(Iteration 29181 / 39200) loss: 1.454953
(Iteration 29191 / 39200) loss: 1.206604
(Iteration 29201 / 39200) loss: 1.552120
(Iteration 29211 / 39200) loss: 1.740137
(Iteration 29221 / 39200) loss: 1.758772
(Iteration 29231 / 39200) loss: 1.449341
(Iteration 29241 / 39200) loss: 1.739700
(Iteration 29251 / 39200) loss: 1.673382
(Iteration 29261 / 39200) loss: 0.995760
(Iteration 29271 / 39200) loss: 1.515915
(Iteration 29281 / 39200) loss: 1.850922
(Iteration 29291 / 39200) loss: 1.429424
(Iteration 29301 / 39200) loss: 1.677995
(Iteration 29311 / 39200) loss: 1.422875
(Iteration 29321 / 39200) loss: 1.602628
(Iteration 29331 / 39200) loss: 1.456064
(Iteration 29341 / 39200) loss: 1.422643
(Iteration 29351 / 39200) loss: 1.455646
(Iteration 29361 / 39200) loss: 1.217668
(Iteration 29371 / 39200) loss: 2.085035
(Iteration 29381 / 39200) loss: 1.656990
(Iteration 29391 / 39200) loss: 1.198083
(Epoch 15 / 20) train acc: 0.651000; val_acc: 0.500000
(Iteration 29401 / 39200) loss: 1.268170
(Iteration 29411 / 39200) loss: 1.561234
```

```
(Iteration 29421 / 39200) loss: 1.463061
(Iteration 29431 / 39200) loss: 1.636666
(Iteration 29441 / 39200) loss: 1.811824
(Iteration 29451 / 39200) loss: 1.529618
(Iteration 29461 / 39200) loss: 1.915115
(Iteration 29471 / 39200) loss: 1.519071
(Iteration 29481 / 39200) loss: 1.360129
(Iteration 29491 / 39200) loss: 1.958800
(Iteration 29501 / 39200) loss: 1.557032
(Iteration 29511 / 39200) loss: 1.651021
(Iteration 29521 / 39200) loss: 1.132722
(Iteration 29531 / 39200) loss: 1.806826
(Iteration 29541 / 39200) loss: 1.306971
(Iteration 29551 / 39200) loss: 1.836648
(Iteration 29561 / 39200) loss: 1.417699
(Iteration 29571 / 39200) loss: 1.410844
(Iteration 29581 / 39200) loss: 1.506653
(Iteration 29591 / 39200) loss: 1.842507
(Iteration 29601 / 39200) loss: 1.513247
(Iteration 29611 / 39200) loss: 1.868621
(Iteration 29621 / 39200) loss: 1.300103
(Iteration 29631 / 39200) loss: 1.614605
(Iteration 29641 / 39200) loss: 1.419024
(Iteration 29651 / 39200) loss: 1.599842
(Iteration 29661 / 39200) loss: 1.679203
(Iteration 29671 / 39200) loss: 1.611803
(Iteration 29681 / 39200) loss: 1.308645
(Iteration 29691 / 39200) loss: 1.613237
(Iteration 29701 / 39200) loss: 1.618107
(Iteration 29711 / 39200) loss: 1.519376
(Iteration 29721 / 39200) loss: 1.194032
(Iteration 29731 / 39200) loss: 1.653390
(Iteration 29741 / 39200) loss: 1.538724
(Iteration 29751 / 39200) loss: 1.484988
(Iteration 29761 / 39200) loss: 1.427362
(Iteration 29771 / 39200) loss: 1.506420
(Iteration 29781 / 39200) loss: 1.564280
(Iteration 29791 / 39200) loss: 1.934096
(Iteration 29801 / 39200) loss: 1.627213
(Iteration 29811 / 39200) loss: 1.421397
(Iteration 29821 / 39200) loss: 1.335993
(Iteration 29831 / 39200) loss: 1.634717
(Iteration 29841 / 39200) loss: 1.379995
(Iteration 29851 / 39200) loss: 1.637062
(Iteration 29861 / 39200) loss: 1.589394
(Iteration 29871 / 39200) loss: 1.197584
(Iteration 29881 / 39200) loss: 1.656763
(Iteration 29891 / 39200) loss: 1.299789
```

```
(Iteration 29901 / 39200) loss: 1.721509
(Iteration 29911 / 39200) loss: 1.338449
(Iteration 29921 / 39200) loss: 1.859346
(Iteration 29931 / 39200) loss: 1.745171
(Iteration 29941 / 39200) loss: 1.608148
(Iteration 29951 / 39200) loss: 1.931545
(Iteration 29961 / 39200) loss: 1.426831
(Iteration 29971 / 39200) loss: 1.482345
(Iteration 29981 / 39200) loss: 1.370031
(Iteration 29991 / 39200) loss: 1.680314
(Iteration 30001 / 39200) loss: 1.394236
(Iteration 30011 / 39200) loss: 1.826143
(Iteration 30021 / 39200) loss: 1.287622
(Iteration 30031 / 39200) loss: 1.691788
(Iteration 30041 / 39200) loss: 1.185041
(Iteration 30051 / 39200) loss: 1.399835
(Iteration 30061 / 39200) loss: 1.438402
(Iteration 30071 / 39200) loss: 1.456896
(Iteration 30081 / 39200) loss: 1.140469
(Iteration 30091 / 39200) loss: 1.769360
(Iteration 30101 / 39200) loss: 1.484734
(Iteration 30111 / 39200) loss: 1.388814
(Iteration 30121 / 39200) loss: 1.316211
(Iteration 30131 / 39200) loss: 1.283785
(Iteration 30141 / 39200) loss: 1.429635
(Iteration 30151 / 39200) loss: 1.601776
(Iteration 30161 / 39200) loss: 1.511722
(Iteration 30171 / 39200) loss: 1.550022
(Iteration 30181 / 39200) loss: 1.685446
(Iteration 30191 / 39200) loss: 1.465046
(Iteration 30201 / 39200) loss: 1.399746
(Iteration 30211 / 39200) loss: 1.544984
(Iteration 30221 / 39200) loss: 1.188306
(Iteration 30231 / 39200) loss: 1.617638
(Iteration 30241 / 39200) loss: 2.181084
(Iteration 30251 / 39200) loss: 1.654965
(Iteration 30261 / 39200) loss: 1.409047
(Iteration 30271 / 39200) loss: 1.781853
(Iteration 30281 / 39200) loss: 1.010434
(Iteration 30291 / 39200) loss: 1.317561
(Iteration 30301 / 39200) loss: 1.550551
(Iteration 30311 / 39200) loss: 1.656199
(Iteration 30321 / 39200) loss: 1.224706
(Iteration 30331 / 39200) loss: 1.349351
(Iteration 30341 / 39200) loss: 1.449005
(Iteration 30351 / 39200) loss: 1.338357
(Iteration 30361 / 39200) loss: 1.209753
(Iteration 30371 / 39200) loss: 1.518863
```

```
(Iteration 30381 / 39200) loss: 1.753399
(Iteration 30391 / 39200) loss: 1.341299
(Iteration 30401 / 39200) loss: 1.706884
(Iteration 30411 / 39200) loss: 1.385562
(Iteration 30421 / 39200) loss: 1.586877
(Iteration 30431 / 39200) loss: 1.292258
(Iteration 30441 / 39200) loss: 1.269171
(Iteration 30451 / 39200) loss: 0.995617
(Iteration 30461 / 39200) loss: 1.454572
(Iteration 30471 / 39200) loss: 1.390917
(Iteration 30481 / 39200) loss: 1.624389
(Iteration 30491 / 39200) loss: 1.328857
(Iteration 30501 / 39200) loss: 1.120433
(Iteration 30511 / 39200) loss: 1.255917
(Iteration 30521 / 39200) loss: 1.275785
(Iteration 30531 / 39200) loss: 1.720295
(Iteration 30541 / 39200) loss: 1.637430
(Iteration 30551 / 39200) loss: 1.316472
(Iteration 30561 / 39200) loss: 1.852382
(Iteration 30571 / 39200) loss: 1.650472
(Iteration 30581 / 39200) loss: 1.179030
(Iteration 30591 / 39200) loss: 1.227076
(Iteration 30601 / 39200) loss: 1.638775
(Iteration 30611 / 39200) loss: 1.586427
(Iteration 30621 / 39200) loss: 1.636596
(Iteration 30631 / 39200) loss: 1.405545
(Iteration 30641 / 39200) loss: 1.759286
(Iteration 30651 / 39200) loss: 1.225711
(Iteration 30661 / 39200) loss: 0.999705
(Iteration 30671 / 39200) loss: 1.262156
(Iteration 30681 / 39200) loss: 1.468125
(Iteration 30691 / 39200) loss: 1.365285
(Iteration 30701 / 39200) loss: 1.450132
(Iteration 30711 / 39200) loss: 1.436479
(Iteration 30721 / 39200) loss: 1.400192
(Iteration 30731 / 39200) loss: 1.603387
(Iteration 30741 / 39200) loss: 1.654280
(Iteration 30751 / 39200) loss: 1.682405
(Iteration 30761 / 39200) loss: 1.524626
(Iteration 30771 / 39200) loss: 1.672687
(Iteration 30781 / 39200) loss: 1.677209
(Iteration 30791 / 39200) loss: 1.290633
(Iteration 30801 / 39200) loss: 1.523992
(Iteration 30811 / 39200) loss: 1.616817
(Iteration 30821 / 39200) loss: 1.399609
(Iteration 30831 / 39200) loss: 1.822575
(Iteration 30841 / 39200) loss: 1.481499
(Iteration 30851 / 39200) loss: 1.538592
```

```
(Iteration 30861 / 39200) loss: 1.593430
(Iteration 30871 / 39200) loss: 1.643304
(Iteration 30881 / 39200) loss: 1.838317
(Iteration 30891 / 39200) loss: 1.651632
(Iteration 30901 / 39200) loss: 1.252317
(Iteration 30911 / 39200) loss: 1.674179
(Iteration 30921 / 39200) loss: 1.201119
(Iteration 30931 / 39200) loss: 1.516833
(Iteration 30941 / 39200) loss: 1.507867
(Iteration 30951 / 39200) loss: 1.507797
(Iteration 30961 / 39200) loss: 1.652100
(Iteration 30971 / 39200) loss: 1.377444
(Iteration 30981 / 39200) loss: 1.144156
(Iteration 30991 / 39200) loss: 1.512694
(Iteration 31001 / 39200) loss: 1.780154
(Iteration 31011 / 39200) loss: 1.600178
(Iteration 31021 / 39200) loss: 1.479761
(Iteration 31031 / 39200) loss: 1.252772
(Iteration 31041 / 39200) loss: 1.290987
(Iteration 31051 / 39200) loss: 1.334451
(Iteration 31061 / 39200) loss: 1.414573
(Iteration 31071 / 39200) loss: 1.582210
(Iteration 31081 / 39200) loss: 1.719518
(Iteration 31091 / 39200) loss: 1.530228
(Iteration 31101 / 39200) loss: 1.245658
(Iteration 31111 / 39200) loss: 1.261052
(Iteration 31121 / 39200) loss: 1.756625
(Iteration 31131 / 39200) loss: 1.447803
(Iteration 31141 / 39200) loss: 1.545256
(Iteration 31151 / 39200) loss: 1.625435
(Iteration 31161 / 39200) loss: 1.546043
(Iteration 31171 / 39200) loss: 1.340828
(Iteration 31181 / 39200) loss: 1.239893
(Iteration 31191 / 39200) loss: 1.683099
(Iteration 31201 / 39200) loss: 1.373051
(Iteration 31211 / 39200) loss: 1.787395
(Iteration 31221 / 39200) loss: 1.485979
(Iteration 31231 / 39200) loss: 1.440255
(Iteration 31241 / 39200) loss: 1.471850
(Iteration 31251 / 39200) loss: 1.530326
(Iteration 31261 / 39200) loss: 2.125432
(Iteration 31271 / 39200) loss: 1.207743
(Iteration 31281 / 39200) loss: 1.721571
(Iteration 31291 / 39200) loss: 1.401531
(Iteration 31301 / 39200) loss: 1.855153
(Iteration 31311 / 39200) loss: 1.498120
(Iteration 31321 / 39200) loss: 1.440713
(Iteration 31331 / 39200) loss: 1.347588
```

```
(Iteration 31341 / 39200) loss: 1.307644
(Iteration 31351 / 39200) loss: 1.684832
(Epoch 16 / 20) train acc: 0.645000; val_acc: 0.482000
(Iteration 31361 / 39200) loss: 1.416704
(Iteration 31371 / 39200) loss: 1.517853
(Iteration 31381 / 39200) loss: 1.413727
(Iteration 31391 / 39200) loss: 1.530921
(Iteration 31401 / 39200) loss: 1.267575
(Iteration 31411 / 39200) loss: 1.474537
(Iteration 31421 / 39200) loss: 1.241049
(Iteration 31431 / 39200) loss: 1.468663
(Iteration 31441 / 39200) loss: 1.669521
(Iteration 31451 / 39200) loss: 1.370465
(Iteration 31461 / 39200) loss: 1.507460
(Iteration 31471 / 39200) loss: 1.097949
(Iteration 31481 / 39200) loss: 1.676609
(Iteration 31491 / 39200) loss: 1.681270
(Iteration 31501 / 39200) loss: 1.364137
(Iteration 31511 / 39200) loss: 1.304947
(Iteration 31521 / 39200) loss: 1.174951
(Iteration 31531 / 39200) loss: 1.479113
(Iteration 31541 / 39200) loss: 1.693407
(Iteration 31551 / 39200) loss: 1.196725
(Iteration 31561 / 39200) loss: 1.550084
(Iteration 31571 / 39200) loss: 1.520920
(Iteration 31581 / 39200) loss: 1.492490
(Iteration 31591 / 39200) loss: 1.218244
(Iteration 31601 / 39200) loss: 1.120983
(Iteration 31611 / 39200) loss: 1.114134
(Iteration 31621 / 39200) loss: 1.524237
(Iteration 31631 / 39200) loss: 1.361641
(Iteration 31641 / 39200) loss: 2.132608
(Iteration 31651 / 39200) loss: 1.760908
(Iteration 31661 / 39200) loss: 1.167751
(Iteration 31671 / 39200) loss: 1.227807
(Iteration 31681 / 39200) loss: 1.493029
(Iteration 31691 / 39200) loss: 1.359665
(Iteration 31701 / 39200) loss: 1.448946
(Iteration 31711 / 39200) loss: 1.423123
(Iteration 31721 / 39200) loss: 1.442334
(Iteration 31731 / 39200) loss: 1.093123
(Iteration 31741 / 39200) loss: 1.345656
(Iteration 31751 / 39200) loss: 1.292351
(Iteration 31761 / 39200) loss: 1.276080
(Iteration 31771 / 39200) loss: 1.513476
(Iteration 31781 / 39200) loss: 1.380712
(Iteration 31791 / 39200) loss: 1.294046
(Iteration 31801 / 39200) loss: 1.617769
```

```
(Iteration 31811 / 39200) loss: 1.473770
(Iteration 31821 / 39200) loss: 1.233139
(Iteration 31831 / 39200) loss: 1.730023
(Iteration 31841 / 39200) loss: 1.828287
(Iteration 31851 / 39200) loss: 1.669301
(Iteration 31861 / 39200) loss: 1.419393
(Iteration 31871 / 39200) loss: 1.639703
(Iteration 31881 / 39200) loss: 1.249335
(Iteration 31891 / 39200) loss: 1.380672
(Iteration 31901 / 39200) loss: 1.482633
(Iteration 31911 / 39200) loss: 1.237726
(Iteration 31921 / 39200) loss: 1.160595
(Iteration 31931 / 39200) loss: 1.607335
(Iteration 31941 / 39200) loss: 1.490485
(Iteration 31951 / 39200) loss: 1.381169
(Iteration 31961 / 39200) loss: 1.431915
(Iteration 31971 / 39200) loss: 1.495456
(Iteration 31981 / 39200) loss: 1.792916
(Iteration 31991 / 39200) loss: 1.727059
(Iteration 32001 / 39200) loss: 1.409067
(Iteration 32011 / 39200) loss: 1.226436
(Iteration 32021 / 39200) loss: 1.374862
(Iteration 32031 / 39200) loss: 1.703820
(Iteration 32041 / 39200) loss: 1.503790
(Iteration 32051 / 39200) loss: 1.484470
(Iteration 32061 / 39200) loss: 1.572349
(Iteration 32071 / 39200) loss: 1.564633
(Iteration 32081 / 39200) loss: 1.277422
(Iteration 32091 / 39200) loss: 1.193977
(Iteration 32101 / 39200) loss: 1.782467
(Iteration 32111 / 39200) loss: 1.354858
(Iteration 32121 / 39200) loss: 1.501407
(Iteration 32131 / 39200) loss: 1.535491
(Iteration 32141 / 39200) loss: 1.728574
(Iteration 32151 / 39200) loss: 1.573236
(Iteration 32161 / 39200) loss: 1.508800
(Iteration 32171 / 39200) loss: 1.575469
(Iteration 32181 / 39200) loss: 1.037228
(Iteration 32191 / 39200) loss: 1.337571
(Iteration 32201 / 39200) loss: 1.000949
(Iteration 32211 / 39200) loss: 1.502224
(Iteration 32221 / 39200) loss: 1.360831
(Iteration 32231 / 39200) loss: 1.391352
(Iteration 32241 / 39200) loss: 1.048774
(Iteration 32251 / 39200) loss: 1.468742
(Iteration 32261 / 39200) loss: 1.557596
(Iteration 32271 / 39200) loss: 1.768791
(Iteration 32281 / 39200) loss: 1.379855
```

```
(Iteration 32291 / 39200) loss: 1.506764
(Iteration 32301 / 39200) loss: 1.460603
(Iteration 32311 / 39200) loss: 1.713261
(Iteration 32321 / 39200) loss: 1.364027
(Iteration 32331 / 39200) loss: 1.685608
(Iteration 32341 / 39200) loss: 1.531462
(Iteration 32351 / 39200) loss: 1.481386
(Iteration 32361 / 39200) loss: 1.789304
(Iteration 32371 / 39200) loss: 1.432724
(Iteration 32381 / 39200) loss: 1.292070
(Iteration 32391 / 39200) loss: 1.614778
(Iteration 32401 / 39200) loss: 1.895735
(Iteration 32411 / 39200) loss: 1.334609
(Iteration 32421 / 39200) loss: 1.637688
(Iteration 32431 / 39200) loss: 1.464319
(Iteration 32441 / 39200) loss: 1.369608
(Iteration 32451 / 39200) loss: 1.592071
(Iteration 32461 / 39200) loss: 1.008999
(Iteration 32471 / 39200) loss: 1.722190
(Iteration 32481 / 39200) loss: 1.304406
(Iteration 32491 / 39200) loss: 1.932123
(Iteration 32501 / 39200) loss: 1.433127
(Iteration 32511 / 39200) loss: 1.652960
(Iteration 32521 / 39200) loss: 1.276698
(Iteration 32531 / 39200) loss: 1.139560
(Iteration 32541 / 39200) loss: 1.268780
(Iteration 32551 / 39200) loss: 1.661296
(Iteration 32561 / 39200) loss: 1.455039
(Iteration 32571 / 39200) loss: 1.385388
(Iteration 32581 / 39200) loss: 1.438992
(Iteration 32591 / 39200) loss: 1.304798
(Iteration 32601 / 39200) loss: 1.372195
(Iteration 32611 / 39200) loss: 1.510938
(Iteration 32621 / 39200) loss: 1.578272
(Iteration 32631 / 39200) loss: 1.754786
(Iteration 32641 / 39200) loss: 1.608560
(Iteration 32651 / 39200) loss: 1.296297
(Iteration 32661 / 39200) loss: 1.627823
(Iteration 32671 / 39200) loss: 1.965453
(Iteration 32681 / 39200) loss: 1.560367
(Iteration 32691 / 39200) loss: 1.528646
(Iteration 32701 / 39200) loss: 1.565675
(Iteration 32711 / 39200) loss: 1.769241
(Iteration 32721 / 39200) loss: 1.204088
(Iteration 32731 / 39200) loss: 1.507641
(Iteration 32741 / 39200) loss: 1.360443
(Iteration 32751 / 39200) loss: 1.090079
(Iteration 32761 / 39200) loss: 1.167075
```

```
(Iteration 32771 / 39200) loss: 1.059410
(Iteration 32781 / 39200) loss: 1.295828
(Iteration 32791 / 39200) loss: 1.427294
(Iteration 32801 / 39200) loss: 1.383049
(Iteration 32811 / 39200) loss: 1.508736
(Iteration 32821 / 39200) loss: 1.416563
(Iteration 32831 / 39200) loss: 1.318313
(Iteration 32841 / 39200) loss: 1.620142
(Iteration 32851 / 39200) loss: 1.036744
(Iteration 32861 / 39200) loss: 1.003084
(Iteration 32871 / 39200) loss: 1.729882
(Iteration 32881 / 39200) loss: 1.315101
(Iteration 32891 / 39200) loss: 1.475427
(Iteration 32901 / 39200) loss: 1.240169
(Iteration 32911 / 39200) loss: 1.687250
(Iteration 32921 / 39200) loss: 1.872202
(Iteration 32931 / 39200) loss: 1.766605
(Iteration 32941 / 39200) loss: 1.480919
(Iteration 32951 / 39200) loss: 1.216843
(Iteration 32961 / 39200) loss: 1.475506
(Iteration 32971 / 39200) loss: 1.296865
(Iteration 32981 / 39200) loss: 1.167140
(Iteration 32991 / 39200) loss: 1.357033
(Iteration 33001 / 39200) loss: 0.895199
(Iteration 33011 / 39200) loss: 1.414228
(Iteration 33021 / 39200) loss: 1.408819
(Iteration 33031 / 39200) loss: 1.349068
(Iteration 33041 / 39200) loss: 1.275000
(Iteration 33051 / 39200) loss: 1.543765
(Iteration 33061 / 39200) loss: 1.599273
(Iteration 33071 / 39200) loss: 2.168365
(Iteration 33081 / 39200) loss: 1.860476
(Iteration 33091 / 39200) loss: 1.541717
(Iteration 33101 / 39200) loss: 1.324964
(Iteration 33111 / 39200) loss: 1.354234
(Iteration 33121 / 39200) loss: 1.354483
(Iteration 33131 / 39200) loss: 1.600570
(Iteration 33141 / 39200) loss: 1.507930
(Iteration 33151 / 39200) loss: 1.240271
(Iteration 33161 / 39200) loss: 1.153396
(Iteration 33171 / 39200) loss: 1.467835
(Iteration 33181 / 39200) loss: 1.514164
(Iteration 33191 / 39200) loss: 1.341502
(Iteration 33201 / 39200) loss: 1.564288
(Iteration 33211 / 39200) loss: 1.298804
(Iteration 33221 / 39200) loss: 1.389998
(Iteration 33231 / 39200) loss: 1.042904
(Iteration 33241 / 39200) loss: 1.269525
```

```
(Iteration 33251 / 39200) loss: 1.386455
(Iteration 33261 / 39200) loss: 1.506511
(Iteration 33271 / 39200) loss: 1.429681
(Iteration 33281 / 39200) loss: 1.432704
(Iteration 33291 / 39200) loss: 1.643177
(Iteration 33301 / 39200) loss: 1.303631
(Iteration 33311 / 39200) loss: 1.344265
(Epoch 17 / 20) train acc: 0.677000; val_acc: 0.509000
(Iteration 33321 / 39200) loss: 1.701588
(Iteration 33331 / 39200) loss: 1.357353
(Iteration 33341 / 39200) loss: 1.456998
(Iteration 33351 / 39200) loss: 1.395586
(Iteration 33361 / 39200) loss: 1.422085
(Iteration 33371 / 39200) loss: 1.576469
(Iteration 33381 / 39200) loss: 1.120021
(Iteration 33391 / 39200) loss: 1.247220
(Iteration 33401 / 39200) loss: 1.384518
(Iteration 33411 / 39200) loss: 1.779992
(Iteration 33421 / 39200) loss: 1.293644
(Iteration 33431 / 39200) loss: 1.251608
(Iteration 33441 / 39200) loss: 1.760409
(Iteration 33451 / 39200) loss: 1.572035
(Iteration 33461 / 39200) loss: 1.432587
(Iteration 33471 / 39200) loss: 1.269798
(Iteration 33481 / 39200) loss: 1.480303
(Iteration 33491 / 39200) loss: 1.465653
(Iteration 33501 / 39200) loss: 1.353290
(Iteration 33511 / 39200) loss: 1.699874
(Iteration 33521 / 39200) loss: 1.480080
(Iteration 33531 / 39200) loss: 1.831197
(Iteration 33541 / 39200) loss: 1.272158
(Iteration 33551 / 39200) loss: 1.219494
(Iteration 33561 / 39200) loss: 1.193065
(Iteration 33571 / 39200) loss: 1.261894
(Iteration 33581 / 39200) loss: 1.600500
(Iteration 33591 / 39200) loss: 1.645236
(Iteration 33601 / 39200) loss: 1.452256
(Iteration 33611 / 39200) loss: 1.444245
(Iteration 33621 / 39200) loss: 1.053107
(Iteration 33631 / 39200) loss: 1.174387
(Iteration 33641 / 39200) loss: 1.405380
(Iteration 33651 / 39200) loss: 1.346238
(Iteration 33661 / 39200) loss: 1.330820
(Iteration 33671 / 39200) loss: 1.031864
(Iteration 33681 / 39200) loss: 1.251780
(Iteration 33691 / 39200) loss: 1.328559
(Iteration 33701 / 39200) loss: 1.655943
(Iteration 33711 / 39200) loss: 1.249956
```

```
(Iteration 33721 / 39200) loss: 1.398031
(Iteration 33731 / 39200) loss: 1.662172
(Iteration 33741 / 39200) loss: 1.717825
(Iteration 33751 / 39200) loss: 1.545189
(Iteration 33761 / 39200) loss: 2.028361
(Iteration 33771 / 39200) loss: 1.625247
(Iteration 33781 / 39200) loss: 1.183814
(Iteration 33791 / 39200) loss: 1.394436
(Iteration 33801 / 39200) loss: 1.719167
(Iteration 33811 / 39200) loss: 1.651305
(Iteration 33821 / 39200) loss: 1.351330
(Iteration 33831 / 39200) loss: 1.472933
(Iteration 33841 / 39200) loss: 1.285661
(Iteration 33851 / 39200) loss: 1.549456
(Iteration 33861 / 39200) loss: 1.625722
(Iteration 33871 / 39200) loss: 1.095486
(Iteration 33881 / 39200) loss: 1.800620
(Iteration 33891 / 39200) loss: 1.150053
(Iteration 33901 / 39200) loss: 1.406783
(Iteration 33911 / 39200) loss: 1.439670
(Iteration 33921 / 39200) loss: 1.182265
(Iteration 33931 / 39200) loss: 1.596299
(Iteration 33941 / 39200) loss: 1.250118
(Iteration 33951 / 39200) loss: 1.580416
(Iteration 33961 / 39200) loss: 1.285870
(Iteration 33971 / 39200) loss: 1.561084
(Iteration 33981 / 39200) loss: 1.361865
(Iteration 33991 / 39200) loss: 1.153302
(Iteration 34001 / 39200) loss: 1.510092
(Iteration 34011 / 39200) loss: 1.289951
(Iteration 34021 / 39200) loss: 1.347326
(Iteration 34031 / 39200) loss: 1.754660
(Iteration 34041 / 39200) loss: 1.353799
(Iteration 34051 / 39200) loss: 1.464031
(Iteration 34061 / 39200) loss: 1.412633
(Iteration 34071 / 39200) loss: 1.196380
(Iteration 34081 / 39200) loss: 1.480189
(Iteration 34091 / 39200) loss: 1.370321
(Iteration 34101 / 39200) loss: 1.151574
(Iteration 34111 / 39200) loss: 1.288494
(Iteration 34121 / 39200) loss: 1.903734
(Iteration 34131 / 39200) loss: 1.632250
(Iteration 34141 / 39200) loss: 1.346707
(Iteration 34151 / 39200) loss: 1.428355
(Iteration 34161 / 39200) loss: 1.450254
(Iteration 34171 / 39200) loss: 1.247845
(Iteration 34181 / 39200) loss: 1.423956
(Iteration 34191 / 39200) loss: 1.169105
```

```
(Iteration 34201 / 39200) loss: 1.293788
(Iteration 34211 / 39200) loss: 1.267634
(Iteration 34221 / 39200) loss: 1.596924
(Iteration 34231 / 39200) loss: 1.401348
(Iteration 34241 / 39200) loss: 1.304712
(Iteration 34251 / 39200) loss: 1.857712
(Iteration 34261 / 39200) loss: 1.366212
(Iteration 34271 / 39200) loss: 1.270783
(Iteration 34281 / 39200) loss: 1.086105
(Iteration 34291 / 39200) loss: 1.231497
(Iteration 34301 / 39200) loss: 1.525250
(Iteration 34311 / 39200) loss: 1.529183
(Iteration 34321 / 39200) loss: 1.345462
(Iteration 34331 / 39200) loss: 1.198653
(Iteration 34341 / 39200) loss: 1.439778
(Iteration 34351 / 39200) loss: 1.514703
(Iteration 34361 / 39200) loss: 1.335170
(Iteration 34371 / 39200) loss: 1.451589
(Iteration 34381 / 39200) loss: 1.352916
(Iteration 34391 / 39200) loss: 1.177386
(Iteration 34401 / 39200) loss: 1.417837
(Iteration 34411 / 39200) loss: 1.380789
(Iteration 34421 / 39200) loss: 1.559795
(Iteration 34431 / 39200) loss: 1.479047
(Iteration 34441 / 39200) loss: 1.428452
(Iteration 34451 / 39200) loss: 1.170994
(Iteration 34461 / 39200) loss: 1.704291
(Iteration 34471 / 39200) loss: 1.274039
(Iteration 34481 / 39200) loss: 1.453364
(Iteration 34491 / 39200) loss: 1.272629
(Iteration 34501 / 39200) loss: 1.492232
(Iteration 34511 / 39200) loss: 1.680927
(Iteration 34521 / 39200) loss: 1.474399
(Iteration 34531 / 39200) loss: 1.369124
(Iteration 34541 / 39200) loss: 1.925900
(Iteration 34551 / 39200) loss: 1.327488
(Iteration 34561 / 39200) loss: 1.440129
(Iteration 34571 / 39200) loss: 1.211037
(Iteration 34581 / 39200) loss: 1.291380
(Iteration 34591 / 39200) loss: 1.376468
(Iteration 34601 / 39200) loss: 1.577378
(Iteration 34611 / 39200) loss: 1.508848
(Iteration 34621 / 39200) loss: 1.102775
(Iteration 34631 / 39200) loss: 1.367920
(Iteration 34641 / 39200) loss: 1.478919
(Iteration 34651 / 39200) loss: 1.323113
(Iteration 34661 / 39200) loss: 1.677404
(Iteration 34671 / 39200) loss: 1.190433
```

```
(Iteration 34681 / 39200) loss: 1.482297
(Iteration 34691 / 39200) loss: 1.630396
(Iteration 34701 / 39200) loss: 1.707793
(Iteration 34711 / 39200) loss: 1.021070
(Iteration 34721 / 39200) loss: 1.143464
(Iteration 34731 / 39200) loss: 1.576517
(Iteration 34741 / 39200) loss: 1.019044
(Iteration 34751 / 39200) loss: 1.495711
(Iteration 34761 / 39200) loss: 1.115346
(Iteration 34771 / 39200) loss: 1.451239
(Iteration 34781 / 39200) loss: 1.450226
(Iteration 34791 / 39200) loss: 0.904220
(Iteration 34801 / 39200) loss: 1.545199
(Iteration 34811 / 39200) loss: 1.306907
(Iteration 34821 / 39200) loss: 1.418194
(Iteration 34831 / 39200) loss: 1.418683
(Iteration 34841 / 39200) loss: 1.551543
(Iteration 34851 / 39200) loss: 1.454566
(Iteration 34861 / 39200) loss: 1.358451
(Iteration 34871 / 39200) loss: 1.664501
(Iteration 34881 / 39200) loss: 1.657102
(Iteration 34891 / 39200) loss: 1.170899
(Iteration 34901 / 39200) loss: 1.419555
(Iteration 34911 / 39200) loss: 1.320450
(Iteration 34921 / 39200) loss: 1.481432
(Iteration 34931 / 39200) loss: 1.031863
(Iteration 34941 / 39200) loss: 1.290981
(Iteration 34951 / 39200) loss: 1.648342
(Iteration 34961 / 39200) loss: 1.171592
(Iteration 34971 / 39200) loss: 1.460604
(Iteration 34981 / 39200) loss: 1.720387
(Iteration 34991 / 39200) loss: 1.391608
(Iteration 35001 / 39200) loss: 1.450929
(Iteration 35011 / 39200) loss: 1.449707
(Iteration 35021 / 39200) loss: 1.537796
(Iteration 35031 / 39200) loss: 1.522576
(Iteration 35041 / 39200) loss: 1.373120
(Iteration 35051 / 39200) loss: 1.710953
(Iteration 35061 / 39200) loss: 1.220398
(Iteration 35071 / 39200) loss: 1.028798
(Iteration 35081 / 39200) loss: 1.298867
(Iteration 35091 / 39200) loss: 1.350449
(Iteration 35101 / 39200) loss: 1.102708
(Iteration 35111 / 39200) loss: 1.417203
(Iteration 35121 / 39200) loss: 1.422151
(Iteration 35131 / 39200) loss: 1.567044
(Iteration 35141 / 39200) loss: 1.535610
(Iteration 35151 / 39200) loss: 1.437203
```

```
(Iteration 35161 / 39200) loss: 1.400968
(Iteration 35171 / 39200) loss: 1.430170
(Iteration 35181 / 39200) loss: 1.172535
(Iteration 35191 / 39200) loss: 1.498970
(Iteration 35201 / 39200) loss: 1.394098
(Iteration 35211 / 39200) loss: 1.259039
(Iteration 35221 / 39200) loss: 1.230124
(Iteration 35231 / 39200) loss: 1.212755
(Iteration 35241 / 39200) loss: 1.666939
(Iteration 35251 / 39200) loss: 1.093462
(Iteration 35261 / 39200) loss: 1.024293
(Iteration 35271 / 39200) loss: 1.727438
(Epoch 18 / 20) train acc: 0.647000; val_acc: 0.500000
(Iteration 35281 / 39200) loss: 1.476981
(Iteration 35291 / 39200) loss: 1.195781
(Iteration 35301 / 39200) loss: 1.502001
(Iteration 35311 / 39200) loss: 1.469574
(Iteration 35321 / 39200) loss: 1.569458
(Iteration 35331 / 39200) loss: 1.181706
(Iteration 35341 / 39200) loss: 1.342331
(Iteration 35351 / 39200) loss: 1.458515
(Iteration 35361 / 39200) loss: 1.293911
(Iteration 35371 / 39200) loss: 1.732477
(Iteration 35381 / 39200) loss: 1.267590
(Iteration 35391 / 39200) loss: 1.386657
(Iteration 35401 / 39200) loss: 1.252087
(Iteration 35411 / 39200) loss: 1.289032
(Iteration 35421 / 39200) loss: 1.269919
(Iteration 35431 / 39200) loss: 1.368044
(Iteration 35441 / 39200) loss: 0.916494
(Iteration 35451 / 39200) loss: 1.443619
(Iteration 35461 / 39200) loss: 1.362490
(Iteration 35471 / 39200) loss: 1.259800
(Iteration 35481 / 39200) loss: 1.007607
(Iteration 35491 / 39200) loss: 0.944435
(Iteration 35501 / 39200) loss: 1.344083
(Iteration 35511 / 39200) loss: 1.463377
(Iteration 35521 / 39200) loss: 1.419801
(Iteration 35531 / 39200) loss: 1.095621
(Iteration 35541 / 39200) loss: 1.436999
(Iteration 35551 / 39200) loss: 1.359796
(Iteration 35561 / 39200) loss: 1.334761
(Iteration 35571 / 39200) loss: 1.091867
(Iteration 35581 / 39200) loss: 1.307941
(Iteration 35591 / 39200) loss: 1.348053
(Iteration 35601 / 39200) loss: 1.212805
(Iteration 35611 / 39200) loss: 1.359728
(Iteration 35621 / 39200) loss: 1.331530
```

```
(Iteration 35631 / 39200) loss: 1.693977
(Iteration 35641 / 39200) loss: 1.752712
(Iteration 35651 / 39200) loss: 1.155541
(Iteration 35661 / 39200) loss: 1.496516
(Iteration 35671 / 39200) loss: 1.304351
(Iteration 35681 / 39200) loss: 1.119535
(Iteration 35691 / 39200) loss: 1.775052
(Iteration 35701 / 39200) loss: 1.613744
(Iteration 35711 / 39200) loss: 0.958307
(Iteration 35721 / 39200) loss: 1.348717
(Iteration 35731 / 39200) loss: 1.133331
(Iteration 35741 / 39200) loss: 1.541133
(Iteration 35751 / 39200) loss: 0.876478
(Iteration 35761 / 39200) loss: 1.129100
(Iteration 35771 / 39200) loss: 1.148091
(Iteration 35781 / 39200) loss: 1.465886
(Iteration 35791 / 39200) loss: 1.612227
(Iteration 35801 / 39200) loss: 1.325527
(Iteration 35811 / 39200) loss: 1.338104
(Iteration 35821 / 39200) loss: 1.340969
(Iteration 35831 / 39200) loss: 0.860067
(Iteration 35841 / 39200) loss: 1.423788
(Iteration 35851 / 39200) loss: 1.452462
(Iteration 35861 / 39200) loss: 1.475106
(Iteration 35871 / 39200) loss: 1.187320
(Iteration 35881 / 39200) loss: 1.210337
(Iteration 35891 / 39200) loss: 1.118186
(Iteration 35901 / 39200) loss: 1.586974
(Iteration 35911 / 39200) loss: 1.279774
(Iteration 35921 / 39200) loss: 1.166296
(Iteration 35931 / 39200) loss: 1.822713
(Iteration 35941 / 39200) loss: 1.328569
(Iteration 35951 / 39200) loss: 1.882089
(Iteration 35961 / 39200) loss: 1.352984
(Iteration 35971 / 39200) loss: 1.663080
(Iteration 35981 / 39200) loss: 1.913983
(Iteration 35991 / 39200) loss: 1.436878
(Iteration 36001 / 39200) loss: 1.332621
(Iteration 36011 / 39200) loss: 1.619652
(Iteration 36021 / 39200) loss: 1.445012
(Iteration 36031 / 39200) loss: 1.532776
(Iteration 36041 / 39200) loss: 1.940010
(Iteration 36051 / 39200) loss: 1.491854
(Iteration 36061 / 39200) loss: 1.405607
(Iteration 36071 / 39200) loss: 1.113199
(Iteration 36081 / 39200) loss: 1.672038
(Iteration 36091 / 39200) loss: 1.618063
(Iteration 36101 / 39200) loss: 1.539236
```

```
(Iteration 36111 / 39200) loss: 1.456146
(Iteration 36121 / 39200) loss: 1.516943
(Iteration 36131 / 39200) loss: 1.268311
(Iteration 36141 / 39200) loss: 1.272073
(Iteration 36151 / 39200) loss: 1.435516
(Iteration 36161 / 39200) loss: 1.071463
(Iteration 36171 / 39200) loss: 1.331917
(Iteration 36181 / 39200) loss: 1.309998
(Iteration 36191 / 39200) loss: 1.043355
(Iteration 36201 / 39200) loss: 1.382366
(Iteration 36211 / 39200) loss: 1.253208
(Iteration 36221 / 39200) loss: 1.605888
(Iteration 36231 / 39200) loss: 1.726504
(Iteration 36241 / 39200) loss: 1.139501
(Iteration 36251 / 39200) loss: 1.520497
(Iteration 36261 / 39200) loss: 1.260281
(Iteration 36271 / 39200) loss: 1.341907
(Iteration 36281 / 39200) loss: 1.508451
(Iteration 36291 / 39200) loss: 1.713773
(Iteration 36301 / 39200) loss: 1.453148
(Iteration 36311 / 39200) loss: 1.172671
(Iteration 36321 / 39200) loss: 0.899352
(Iteration 36331 / 39200) loss: 1.407308
(Iteration 36341 / 39200) loss: 1.634321
(Iteration 36351 / 39200) loss: 1.256451
(Iteration 36361 / 39200) loss: 1.312030
(Iteration 36371 / 39200) loss: 1.340588
(Iteration 36381 / 39200) loss: 1.494265
(Iteration 36391 / 39200) loss: 1.234931
(Iteration 36401 / 39200) loss: 1.458255
(Iteration 36411 / 39200) loss: 1.693504
(Iteration 36421 / 39200) loss: 1.435053
(Iteration 36431 / 39200) loss: 1.451514
(Iteration 36441 / 39200) loss: 1.127711
(Iteration 36451 / 39200) loss: 1.641912
(Iteration 36461 / 39200) loss: 1.043559
(Iteration 36471 / 39200) loss: 1.084850
(Iteration 36481 / 39200) loss: 1.328741
(Iteration 36491 / 39200) loss: 1.070901
(Iteration 36501 / 39200) loss: 1.349900
(Iteration 36511 / 39200) loss: 1.640678
(Iteration 36521 / 39200) loss: 1.245995
(Iteration 36531 / 39200) loss: 1.209620
(Iteration 36541 / 39200) loss: 1.379398
(Iteration 36551 / 39200) loss: 1.145798
(Iteration 36561 / 39200) loss: 1.939534
(Iteration 36571 / 39200) loss: 1.609703
(Iteration 36581 / 39200) loss: 1.708893
```

```
(Iteration 36591 / 39200) loss: 1.569210
(Iteration 36601 / 39200) loss: 1.289036
(Iteration 36611 / 39200) loss: 1.393396
(Iteration 36621 / 39200) loss: 1.563804
(Iteration 36631 / 39200) loss: 1.258412
(Iteration 36641 / 39200) loss: 1.416972
(Iteration 36651 / 39200) loss: 1.290926
(Iteration 36661 / 39200) loss: 1.781513
(Iteration 36671 / 39200) loss: 1.565778
(Iteration 36681 / 39200) loss: 1.872142
(Iteration 36691 / 39200) loss: 1.097919
(Iteration 36701 / 39200) loss: 1.215580
(Iteration 36711 / 39200) loss: 1.225038
(Iteration 36721 / 39200) loss: 1.605864
(Iteration 36731 / 39200) loss: 1.293998
(Iteration 36741 / 39200) loss: 1.765372
(Iteration 36751 / 39200) loss: 1.396844
(Iteration 36761 / 39200) loss: 1.492988
(Iteration 36771 / 39200) loss: 1.545775
(Iteration 36781 / 39200) loss: 1.085537
(Iteration 36791 / 39200) loss: 1.242167
(Iteration 36801 / 39200) loss: 1.421918
(Iteration 36811 / 39200) loss: 1.474324
(Iteration 36821 / 39200) loss: 1.238925
(Iteration 36831 / 39200) loss: 1.542159
(Iteration 36841 / 39200) loss: 1.292354
(Iteration 36851 / 39200) loss: 1.442353
(Iteration 36861 / 39200) loss: 1.134459
(Iteration 36871 / 39200) loss: 1.620004
(Iteration 36881 / 39200) loss: 1.434321
(Iteration 36891 / 39200) loss: 1.337430
(Iteration 36901 / 39200) loss: 1.284091
(Iteration 36911 / 39200) loss: 1.534295
(Iteration 36921 / 39200) loss: 1.844313
(Iteration 36931 / 39200) loss: 1.140643
(Iteration 36941 / 39200) loss: 1.263931
(Iteration 36951 / 39200) loss: 1.439366
(Iteration 36961 / 39200) loss: 1.458978
(Iteration 36971 / 39200) loss: 1.617524
(Iteration 36981 / 39200) loss: 1.213317
(Iteration 36991 / 39200) loss: 0.997463
(Iteration 37001 / 39200) loss: 1.067118
(Iteration 37011 / 39200) loss: 1.245735
(Iteration 37021 / 39200) loss: 1.321416
(Iteration 37031 / 39200) loss: 1.200383
(Iteration 37041 / 39200) loss: 1.205960
(Iteration 37051 / 39200) loss: 1.497934
(Iteration 37061 / 39200) loss: 1.707930
```

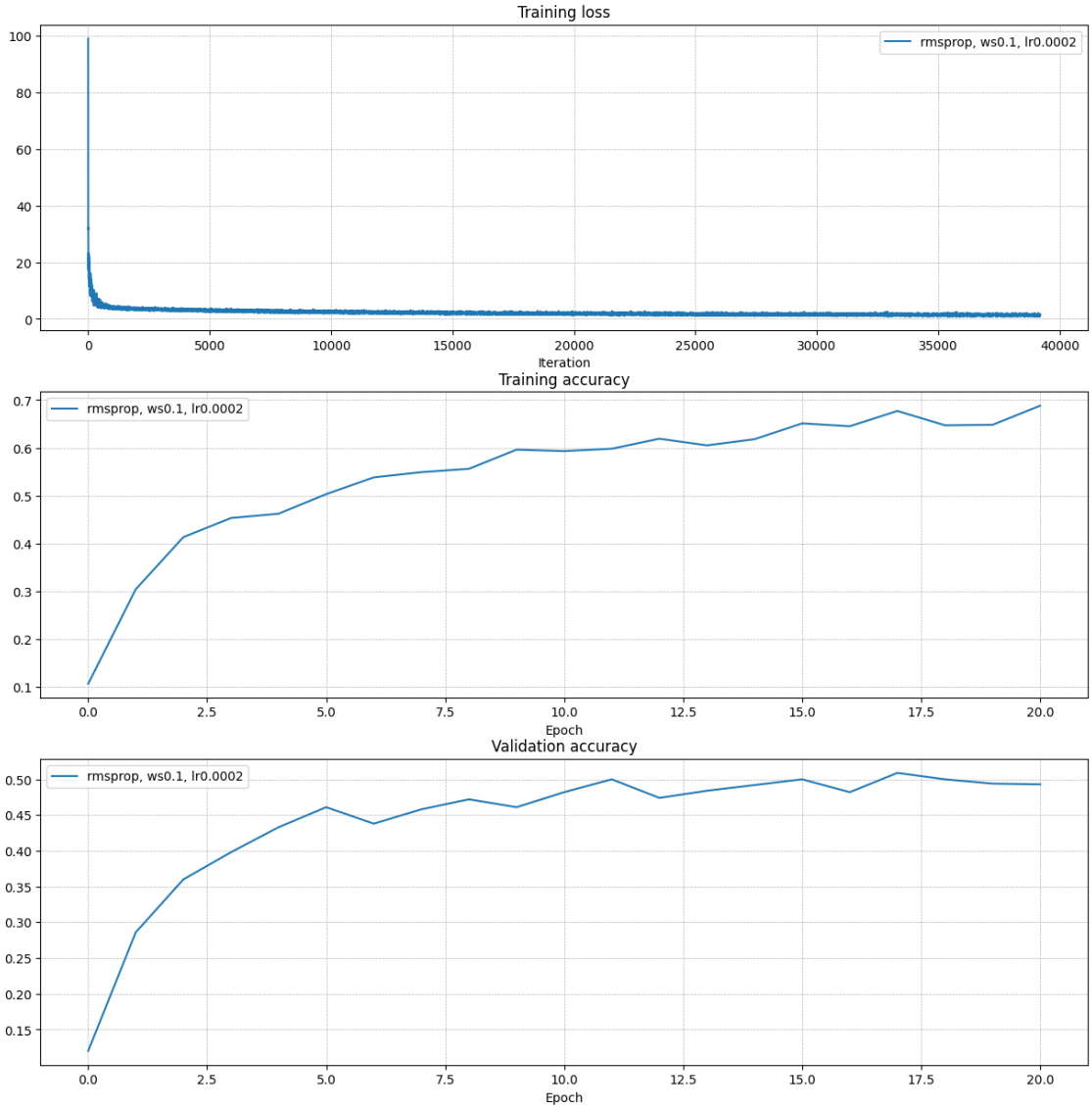
```
(Iteration 37071 / 39200) loss: 1.615444
(Iteration 37081 / 39200) loss: 1.248158
(Iteration 37091 / 39200) loss: 1.752790
(Iteration 37101 / 39200) loss: 1.598281
(Iteration 37111 / 39200) loss: 1.509570
(Iteration 37121 / 39200) loss: 1.537364
(Iteration 37131 / 39200) loss: 1.158959
(Iteration 37141 / 39200) loss: 1.628289
(Iteration 37151 / 39200) loss: 1.069563
(Iteration 37161 / 39200) loss: 1.423459
(Iteration 37171 / 39200) loss: 1.276030
(Iteration 37181 / 39200) loss: 1.134289
(Iteration 37191 / 39200) loss: 1.391873
(Iteration 37201 / 39200) loss: 1.574925
(Iteration 37211 / 39200) loss: 1.387143
(Iteration 37221 / 39200) loss: 1.142199
(Iteration 37231 / 39200) loss: 1.349629
(Epoch 19 / 20) train acc: 0.648000; val_acc: 0.494000
(Iteration 37241 / 39200) loss: 1.249341
(Iteration 37251 / 39200) loss: 1.472885
(Iteration 37261 / 39200) loss: 1.735570
(Iteration 37271 / 39200) loss: 1.311743
(Iteration 37281 / 39200) loss: 1.658445
(Iteration 37291 / 39200) loss: 1.297133
(Iteration 37301 / 39200) loss: 1.051814
(Iteration 37311 / 39200) loss: 1.334065
(Iteration 37321 / 39200) loss: 1.088669
(Iteration 37331 / 39200) loss: 1.366819
(Iteration 37341 / 39200) loss: 1.156204
(Iteration 37351 / 39200) loss: 1.553783
(Iteration 37361 / 39200) loss: 1.298652
(Iteration 37371 / 39200) loss: 1.124581
(Iteration 37381 / 39200) loss: 1.517072
(Iteration 37391 / 39200) loss: 1.275822
(Iteration 37401 / 39200) loss: 1.263263
(Iteration 37411 / 39200) loss: 1.625171
(Iteration 37421 / 39200) loss: 1.756525
(Iteration 37431 / 39200) loss: 1.102872
(Iteration 37441 / 39200) loss: 2.040801
(Iteration 37451 / 39200) loss: 1.398412
(Iteration 37461 / 39200) loss: 1.200263
(Iteration 37471 / 39200) loss: 1.551652
(Iteration 37481 / 39200) loss: 1.430313
(Iteration 37491 / 39200) loss: 1.372929
(Iteration 37501 / 39200) loss: 1.467829
(Iteration 37511 / 39200) loss: 1.505178
(Iteration 37521 / 39200) loss: 1.567284
(Iteration 37531 / 39200) loss: 1.080982
```

```
(Iteration 37541 / 39200) loss: 1.060541
(Iteration 37551 / 39200) loss: 1.309907
(Iteration 37561 / 39200) loss: 1.576422
(Iteration 37571 / 39200) loss: 1.228638
(Iteration 37581 / 39200) loss: 1.299896
(Iteration 37591 / 39200) loss: 1.185022
(Iteration 37601 / 39200) loss: 1.470189
(Iteration 37611 / 39200) loss: 1.143546
(Iteration 37621 / 39200) loss: 1.191052
(Iteration 37631 / 39200) loss: 1.131124
(Iteration 37641 / 39200) loss: 1.266842
(Iteration 37651 / 39200) loss: 1.190538
(Iteration 37661 / 39200) loss: 1.417906
(Iteration 37671 / 39200) loss: 1.623015
(Iteration 37681 / 39200) loss: 1.806519
(Iteration 37691 / 39200) loss: 1.440757
(Iteration 37701 / 39200) loss: 1.211024
(Iteration 37711 / 39200) loss: 1.376554
(Iteration 37721 / 39200) loss: 1.241675
(Iteration 37731 / 39200) loss: 1.192668
(Iteration 37741 / 39200) loss: 1.491416
(Iteration 37751 / 39200) loss: 1.335424
(Iteration 37761 / 39200) loss: 1.202900
(Iteration 37771 / 39200) loss: 1.267264
(Iteration 37781 / 39200) loss: 0.978754
(Iteration 37791 / 39200) loss: 1.454704
(Iteration 37801 / 39200) loss: 1.392530
(Iteration 37811 / 39200) loss: 1.476465
(Iteration 37821 / 39200) loss: 1.470208
(Iteration 37831 / 39200) loss: 1.293918
(Iteration 37841 / 39200) loss: 1.149473
(Iteration 37851 / 39200) loss: 1.561985
(Iteration 37861 / 39200) loss: 1.367547
(Iteration 37871 / 39200) loss: 1.290144
(Iteration 37881 / 39200) loss: 1.564326
(Iteration 37891 / 39200) loss: 1.475558
(Iteration 37901 / 39200) loss: 1.410196
(Iteration 37911 / 39200) loss: 1.519216
(Iteration 37921 / 39200) loss: 0.973639
(Iteration 37931 / 39200) loss: 1.072300
(Iteration 37941 / 39200) loss: 1.299453
(Iteration 37951 / 39200) loss: 1.241131
(Iteration 37961 / 39200) loss: 1.410503
(Iteration 37971 / 39200) loss: 0.941342
(Iteration 37981 / 39200) loss: 1.447829
(Iteration 37991 / 39200) loss: 1.400686
(Iteration 38001 / 39200) loss: 1.424159
(Iteration 38011 / 39200) loss: 1.205815
```

```
(Iteration 38021 / 39200) loss: 1.293946
(Iteration 38031 / 39200) loss: 1.173511
(Iteration 38041 / 39200) loss: 0.971503
(Iteration 38051 / 39200) loss: 1.197984
(Iteration 38061 / 39200) loss: 0.978558
(Iteration 38071 / 39200) loss: 1.401334
(Iteration 38081 / 39200) loss: 1.485521
(Iteration 38091 / 39200) loss: 1.665082
(Iteration 38101 / 39200) loss: 1.435251
(Iteration 38111 / 39200) loss: 1.387900
(Iteration 38121 / 39200) loss: 1.165589
(Iteration 38131 / 39200) loss: 0.972649
(Iteration 38141 / 39200) loss: 1.440596
(Iteration 38151 / 39200) loss: 1.246195
(Iteration 38161 / 39200) loss: 1.189853
(Iteration 38171 / 39200) loss: 1.273057
(Iteration 38181 / 39200) loss: 1.211891
(Iteration 38191 / 39200) loss: 1.169306
(Iteration 38201 / 39200) loss: 1.582533
(Iteration 38211 / 39200) loss: 1.142336
(Iteration 38221 / 39200) loss: 1.187238
(Iteration 38231 / 39200) loss: 1.588845
(Iteration 38241 / 39200) loss: 1.369621
(Iteration 38251 / 39200) loss: 1.023081
(Iteration 38261 / 39200) loss: 1.435029
(Iteration 38271 / 39200) loss: 1.570801
(Iteration 38281 / 39200) loss: 1.602759
(Iteration 38291 / 39200) loss: 0.984828
(Iteration 38301 / 39200) loss: 1.642299
(Iteration 38311 / 39200) loss: 1.527003
(Iteration 38321 / 39200) loss: 1.541091
(Iteration 38331 / 39200) loss: 1.167358
(Iteration 38341 / 39200) loss: 1.014580
(Iteration 38351 / 39200) loss: 1.141879
(Iteration 38361 / 39200) loss: 1.427497
(Iteration 38371 / 39200) loss: 1.966043
(Iteration 38381 / 39200) loss: 1.231364
(Iteration 38391 / 39200) loss: 1.282059
(Iteration 38401 / 39200) loss: 1.160552
(Iteration 38411 / 39200) loss: 0.978298
(Iteration 38421 / 39200) loss: 1.655309
(Iteration 38431 / 39200) loss: 1.527084
(Iteration 38441 / 39200) loss: 1.111299
(Iteration 38451 / 39200) loss: 1.574351
(Iteration 38461 / 39200) loss: 1.305246
(Iteration 38471 / 39200) loss: 1.853706
(Iteration 38481 / 39200) loss: 1.283563
(Iteration 38491 / 39200) loss: 1.675225
```

```
(Iteration 38501 / 39200) loss: 1.560568
(Iteration 38511 / 39200) loss: 0.990923
(Iteration 38521 / 39200) loss: 1.485380
(Iteration 38531 / 39200) loss: 1.271573
(Iteration 38541 / 39200) loss: 1.148024
(Iteration 38551 / 39200) loss: 1.271971
(Iteration 38561 / 39200) loss: 1.489373
(Iteration 38571 / 39200) loss: 1.354888
(Iteration 38581 / 39200) loss: 1.682079
(Iteration 38591 / 39200) loss: 1.606218
(Iteration 38601 / 39200) loss: 1.644419
(Iteration 38611 / 39200) loss: 1.003604
(Iteration 38621 / 39200) loss: 1.348679
(Iteration 38631 / 39200) loss: 1.017322
(Iteration 38641 / 39200) loss: 1.566323
(Iteration 38651 / 39200) loss: 1.312563
(Iteration 38661 / 39200) loss: 1.312507
(Iteration 38671 / 39200) loss: 1.687044
(Iteration 38681 / 39200) loss: 1.276844
(Iteration 38691 / 39200) loss: 1.376110
(Iteration 38701 / 39200) loss: 1.914534
(Iteration 38711 / 39200) loss: 1.308090
(Iteration 38721 / 39200) loss: 1.211445
(Iteration 38731 / 39200) loss: 1.412544
(Iteration 38741 / 39200) loss: 1.026652
(Iteration 38751 / 39200) loss: 1.422744
(Iteration 38761 / 39200) loss: 1.695596
(Iteration 38771 / 39200) loss: 1.437381
(Iteration 38781 / 39200) loss: 1.558361
(Iteration 38791 / 39200) loss: 1.407354
(Iteration 38801 / 39200) loss: 1.064089
(Iteration 38811 / 39200) loss: 1.238077
(Iteration 38821 / 39200) loss: 1.354977
(Iteration 38831 / 39200) loss: 1.090291
(Iteration 38841 / 39200) loss: 0.972932
(Iteration 38851 / 39200) loss: 1.290926
(Iteration 38861 / 39200) loss: 1.151526
(Iteration 38871 / 39200) loss: 1.246231
(Iteration 38881 / 39200) loss: 1.074610
(Iteration 38891 / 39200) loss: 1.349137
(Iteration 38901 / 39200) loss: 1.107125
(Iteration 38911 / 39200) loss: 0.998923
(Iteration 38921 / 39200) loss: 1.092930
(Iteration 38931 / 39200) loss: 0.984845
(Iteration 38941 / 39200) loss: 1.615688
(Iteration 38951 / 39200) loss: 1.380825
(Iteration 38961 / 39200) loss: 2.012066
(Iteration 38971 / 39200) loss: 1.008976
```

```
(Iteration 38981 / 39200) loss: 1.245417
(Iteration 38991 / 39200) loss: 1.014532
(Iteration 39001 / 39200) loss: 1.261342
(Iteration 39011 / 39200) loss: 1.152411
(Iteration 39021 / 39200) loss: 1.592623
(Iteration 39031 / 39200) loss: 1.061792
(Iteration 39041 / 39200) loss: 1.358806
(Iteration 39051 / 39200) loss: 1.519545
(Iteration 39061 / 39200) loss: 1.351524
(Iteration 39071 / 39200) loss: 1.483020
(Iteration 39081 / 39200) loss: 1.179015
(Iteration 39091 / 39200) loss: 1.280756
(Iteration 39101 / 39200) loss: 1.219732
(Iteration 39111 / 39200) loss: 1.384701
(Iteration 39121 / 39200) loss: 1.506303
(Iteration 39131 / 39200) loss: 1.068632
(Iteration 39141 / 39200) loss: 1.128251
(Iteration 39151 / 39200) loss: 1.094524
(Iteration 39161 / 39200) loss: 1.210414
(Iteration 39171 / 39200) loss: 1.395331
(Iteration 39181 / 39200) loss: 1.557882
(Iteration 39191 / 39200) loss: 1.214970
(Epoch 20 / 20) train acc: 0.688000; val_acc: 0.493000
weight_scale  1.000000e-01 lr 2.000000e-04 rule rmsprop train accuracy: 0.688000
val accuracy: 0.509000
best val accuracy achieved: 0.509000
weight_scale  1.000000e-01 lr 2.000000e-04 rule rmsprop train accuracy: 0.688000
val accuracy: 0.509000
best val accuracy achieved: 0.509000
```



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.

[20]:
```
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:  0.509
Test set accuracy:  0.519
```