Convolutional neural networks

In this notebook, we'll put together our convolutional layers to implement a 3-layer CNN. Then, we'll ask you to implement a CNN that can achieve > 65% validation error on CIFAR-10.

CS231n has built a solid API for building these modular frameworks and training them, and we will use their very well implemented framework as opposed to "reinventing the wheel." This includes using their Solver, various utility functions, their layer structure, and their implementation of fast CNN layers. This also includes nndl.fc_net, nndl.layers, and nndl.layer_utils. As in prior assignments, we thank Serena Yeung & Justin Johnson for permission to use code written for the CS 231n class (cs231n.stanford.edu).

If you have not completed the Spatial BatchNorm Notebook, please see the following description from that notebook:

Please copy and paste your prior implemented code from HW #4 to start this assignment. If you did not correctly implement the layers in HW #4, you may collaborate with a classmate to use their layer implementations from HW #4. You may also visit TA or Prof OH to correct your implementation.

You'll want to copy and paste from HW #4:

```
- layers.py for your FC network layers, as well as batchnorm and dropout.
- layer_utils.py for your combined FC network layers.
- optim.py for your optimizers.
```

Be sure to place these in the `nndl/` directory so they're imported correctly. Note, as announced in class, we will not be releasing our solutions.

In [1]:
```
# As usual, a bit of setup

import numpy as np
import matplotlib.pyplot as plt
from nndl.cnn import *
from cs231n.data_utils import get_CIFAR10_data
from cs231n.gradient_check import eval_numerical_gradient_array, eval_
numerical_gradient
from nndl.layers import *
from nndl.conv_layers import *
from cs231n.fast_layers import *
from cs231n.solver import Solver

%matplotlib inline
plt.rcParams['figure.figsize'] = (10.0, 8.0) # set default size of plo
ts
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
))))
```

In [2]:
```
# Load the (preprocessed) CIFAR10 data.

data = get_CIFAR10_data()
for k in data.keys():
  print('{}: {} '.format(k, data[k].shape))
```

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

Three layer CNN

In this notebook, you will implement a three layer CNN. The `ThreeLayerConvNet` class is in `nndl/cnn.py`. You'll need to modify that code for this section, including the initialization, as well as the calculation of the loss and gradients. You should be able to use the building blocks you have either earlier coded or that we have provided. Be sure to use the fast layers.

The architecture of this CNN will be:

conv - relu - 2x2 max pool - affine - relu - affine - softmax

We won't use batchnorm yet. You've also done enough of these to know how to debug; use the cells below.

Note: As we are implementing several layers CNN networks. The gradient error can be expected for the `eval_numerical_gradient()` function. If your `W1 max relative error` and `W2 max relative error` are around or below 0.01, they should be acceptable. Other errors should be less than 1e-5.

In [10]:
```
num_inputs = 2
input_dim = (3, 16, 16)
reg = 0.0
num_classes = 10
X = np.random.randn(num_inputs, *input_dim)
y = np.random.randint(num_classes, size=num_inputs)

model = ThreeLayerConvNet(num_filters=3, filter_size=3,
                          input_dim=input_dim, hidden_dim=7,
                          dtype=np.float64)
loss, grads = model.loss(X, y)
for param_name in sorted(grads):
    f = lambda _: model.loss(X, y)[0]
    param_grad_num = eval_numerical_gradient(f, model.params[param_nam
e], verbose=False, h=1e-6)
    e = rel_error(param_grad_num, grads[param_name])
    print('{} max relative error: {}'.format(param_name, rel_error(par
am_grad_num, grads[param_name])))
```

```
W1 max relative error: 0.003339638093857734
W2 max relative error: 0.007733551671560727
W3 max relative error: 0.00013319376741324182
b1 max relative error: 2.912627580018027e-06
b2 max relative error: 6.225815960689804e-07
b3 max relative error: 8.210797326525663e-10
```

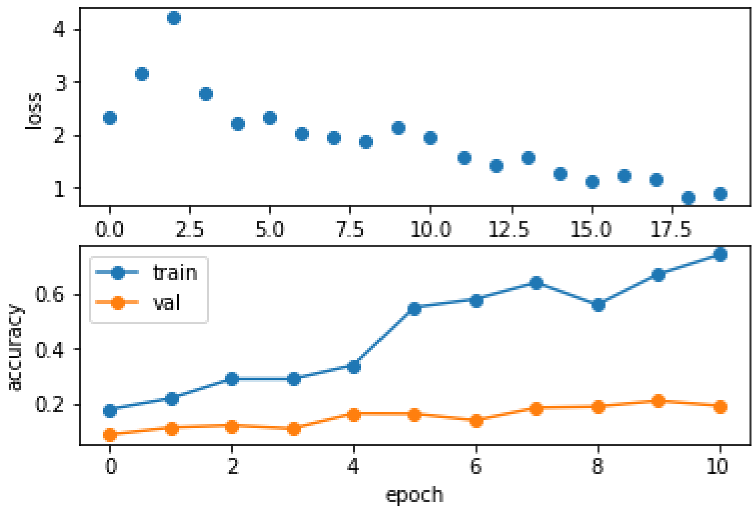
Overfit small dataset

To check your CNN implementation, let's overfit a small dataset.

In [11]:
```
num_train = 100
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'],
}

model = ThreeLayerConvNet(weight_scale=1e-2)

solver = Solver(model, small_data,
                num_epochs=10, batch_size=50,
                update_rule='adam',
                optim_config={
                  'learning_rate': 1e-3,
                },
                verbose=True, print_every=1)
solver.train()
```

```
(Iteration 1 / 20) loss: 2.338184
(Epoch 0 / 10) train acc: 0.180000; val_acc: 0.088000
(Iteration 2 / 20) loss: 3.145421
(Epoch 1 / 10) train acc: 0.220000; val_acc: 0.114000
(Iteration 3 / 20) loss: 4.220575
(Iteration 4 / 20) loss: 2.789140
(Epoch 2 / 10) train acc: 0.290000; val_acc: 0.122000
(Iteration 5 / 20) loss: 2.198183
(Iteration 6 / 20) loss: 2.334225
(Epoch 3 / 10) train acc: 0.290000; val_acc: 0.110000
(Iteration 7 / 20) loss: 2.026942
(Iteration 8 / 20) loss: 1.948826
(Epoch 4 / 10) train acc: 0.340000; val_acc: 0.165000
(Iteration 9 / 20) loss: 1.859220
(Iteration 10 / 20) loss: 2.146719
(Epoch 5 / 10) train acc: 0.550000; val_acc: 0.164000
(Iteration 11 / 20) loss: 1.968799
(Iteration 12 / 20) loss: 1.587138
(Epoch 6 / 10) train acc: 0.580000; val_acc: 0.140000
(Iteration 13 / 20) loss: 1.415540
(Iteration 14 / 20) loss: 1.577199
(Epoch 7 / 10) train acc: 0.640000; val_acc: 0.186000
(Iteration 15 / 20) loss: 1.260498
(Iteration 16 / 20) loss: 1.126015
(Epoch 8 / 10) train acc: 0.560000; val_acc: 0.190000
(Iteration 17 / 20) loss: 1.230003
(Iteration 18 / 20) loss: 1.154813
(Epoch 9 / 10) train acc: 0.670000; val_acc: 0.211000
(Iteration 19 / 20) loss: 0.833167
(Iteration 20 / 20) loss: 0.915641
(Epoch 10 / 10) train acc: 0.740000; val_acc: 0.193000
```

In [12]:
```
plt.subplot(2, 1, 1)
plt.plot(solver.loss_history, 'o')
plt.xlabel('iteration')
plt.ylabel('loss')

plt.subplot(2, 1, 2)
plt.plot(solver.train_acc_history, '-o')
plt.plot(solver.val_acc_history, '-o')
plt.legend(['train', 'val'], loc='upper left')
plt.xlabel('epoch')
plt.ylabel('accuracy')
plt.show()
```



Train the network

Now we train the 3 layer CNN on CIFAR-10 and assess its accuracy.

In [13]:
```
model = ThreeLayerConvNet(weight_scale=0.001, hidden_dim=500, reg=0.00
1)

solver = Solver(model, data,
                num_epochs=1, batch_size=50,
                update_rule='adam',
                optim_config={
                  'learning_rate': 1e-3,
                },
                verbose=True, print_every=20)
solver.train()
```

```
(Iteration 1 / 980) loss: 2.302710
(Epoch 0 / 1) train acc: 0.099000; val_acc: 0.107000
(Iteration 21 / 980) loss: 1.867889
(Iteration 41 / 980) loss: 1.823136
(Iteration 61 / 980) loss: 1.759033
(Iteration 81 / 980) loss: 1.863504
(Iteration 101 / 980) loss: 1.882208
(Iteration 121 / 980) loss: 1.974029
(Iteration 141 / 980) loss: 1.997073
(Iteration 161 / 980) loss: 1.563230
(Iteration 181 / 980) loss: 1.585403
(Iteration 201 / 980) loss: 1.759488
(Iteration 221 / 980) loss: 1.753828
(Iteration 241 / 980) loss: 2.131299
(Iteration 261 / 980) loss: 1.408404
(Iteration 281 / 980) loss: 1.405383
(Iteration 301 / 980) loss: 1.720499
(Iteration 321 / 980) loss: 1.567853
(Iteration 341 / 980) loss: 1.481917
(Iteration 361 / 980) loss: 1.491857
(Iteration 381 / 980) loss: 1.694706
(Iteration 401 / 980) loss: 1.911365
(Iteration 421 / 980) loss: 1.884573
(Iteration 441 / 980) loss: 1.511824
(Iteration 461 / 980) loss: 1.810901
(Iteration 481 / 980) loss: 1.654326
(Iteration 501 / 980) loss: 1.414904
(Iteration 521 / 980) loss: 1.754829
(Iteration 541 / 980) loss: 1.792938
(Iteration 561 / 980) loss: 1.961524
(Iteration 581 / 980) loss: 1.524744
(Iteration 601 / 980) loss: 1.466191
(Iteration 621 / 980) loss: 1.757177
(Iteration 641 / 980) loss: 1.750617
(Iteration 661 / 980) loss: 1.822015
(Iteration 681 / 980) loss: 1.608533
(Iteration 701 / 980) loss: 1.471488
(Iteration 721 / 980) loss: 1.617726
(Iteration 741 / 980) loss: 1.343086
(Iteration 761 / 980) loss: 1.537552
(Iteration 781 / 980) loss: 1.539328
(Iteration 801 / 980) loss: 1.431361
(Iteration 821 / 980) loss: 1.492462
(Iteration 841 / 980) loss: 1.674484
(Iteration 861 / 980) loss: 1.634723
(Iteration 881 / 980) loss: 1.608059
(Iteration 901 / 980) loss: 1.704482
(Iteration 921 / 980) loss: 1.465134
(Iteration 941 / 980) loss: 1.906660
(Iteration 961 / 980) loss: 1.723571
(Epoch 1 / 1) train acc: 0.448000; val_acc: 0.456000
```

Get > 65% validation accuracy on CIFAR-10.

In the last part of the assignment, we'll now ask you to train a CNN to get better than 65% validation accuracy on CIFAR-10.

Things you should try:

- Filter size: Above we used 7x7; but VGGNet and onwards showed stacks of 3x3 filters are good.
- Number of filters: Above we used 32 filters. Do more or fewer do better?
- Batch normalization: Try adding spatial batch normalization after convolution layers and vanilla batch normalization after affine layers. Do your networks train faster?
- Network architecture: Can a deeper CNN do better? Consider these architectures:
  - [conv-relu-pool]xN - conv - relu - [affine]xM - [softmax or SVM]
  - [conv-relu-pool]XN - [affine]XM - [softmax or SVM]
  - [conv-relu-conv-relu-pool]xN - [affine]xM - [softmax or SVM]

Tips for training

For each network architecture that you try, you should tune the learning rate and regularization strength. When doing this there are a couple of important things to keep in mind:

- If the parameters are working well, you should see improvement within a few hundred iterations
- Remember the coarse-to-fine approach for hyperparameter tuning: start by testing a large range of hyperparameters for just a few training iterations to find the combinations of parameters that are working at all.
- Once you have found some sets of parameters that seem to work, search more finely around these parameters. You may need to train for more epochs.

In [16]:
```
# =============================================================== #
# YOUR CODE HERE:
#   Implement a CNN to achieve greater than 65% validation accuracy
#   on CIFAR-10.
# =============================================================== #

model = ThreeLayerConvNet(weight_scale=0.001, hidden_dim=800, reg=0.01
, num_filters=80, filter_size=3)

solver = Solver(model, data,
                num_epochs=15, batch_size=128,
                update_rule='adam',
                optim_config={
                  'learning_rate': 6e-4,
                },
                verbose=True, print_every=20)
solver.train()

# =============================================================== #
# END YOUR CODE HERE
# =============================================================== #
```

```
(Iteration 1 / 5730) loss: 2.302662
(Epoch 0 / 15) train acc: 0.100000; val_acc: 0.113000
(Iteration 21 / 5730) loss: 1.822297
(Iteration 41 / 5730) loss: 1.739436
(Iteration 61 / 5730) loss: 1.697439
(Iteration 81 / 5730) loss: 1.636978
(Iteration 101 / 5730) loss: 1.600068
(Iteration 121 / 5730) loss: 1.464973
(Iteration 141 / 5730) loss: 1.323235
(Iteration 161 / 5730) loss: 1.508464
(Iteration 181 / 5730) loss: 1.562103
(Iteration 201 / 5730) loss: 1.697535
(Iteration 221 / 5730) loss: 1.382375
(Iteration 241 / 5730) loss: 1.271008
(Iteration 261 / 5730) loss: 1.375135
(Iteration 281 / 5730) loss: 1.275878
(Iteration 301 / 5730) loss: 1.513031
(Iteration 321 / 5730) loss: 1.395120
(Iteration 341 / 5730) loss: 1.417776
(Iteration 361 / 5730) loss: 1.199762
(Iteration 381 / 5730) loss: 1.279455
(Epoch 1 / 15) train acc: 0.558000; val_acc: 0.565000
(Iteration 401 / 5730) loss: 1.220100
(Iteration 421 / 5730) loss: 1.225320
(Iteration 441 / 5730) loss: 1.278838
(Iteration 461 / 5730) loss: 1.276752
(Iteration 481 / 5730) loss: 1.307980
(Iteration 501 / 5730) loss: 1.218306
(Iteration 521 / 5730) loss: 1.165487
(Iteration 541 / 5730) loss: 1.083059
(Iteration 561 / 5730) loss: 1.273285
(Iteration 581 / 5730) loss: 1.119473
(Iteration 601 / 5730) loss: 1.335253
(Iteration 621 / 5730) loss: 1.070652
(Iteration 641 / 5730) loss: 1.318953
(Iteration 661 / 5730) loss: 1.097223
(Iteration 681 / 5730) loss: 1.294324
(Iteration 701 / 5730) loss: 1.118972
(Iteration 721 / 5730) loss: 1.143132
(Iteration 741 / 5730) loss: 1.135419
(Iteration 761 / 5730) loss: 0.984195
(Epoch 2 / 15) train acc: 0.615000; val_acc: 0.577000
(Iteration 781 / 5730) loss: 1.109513
(Iteration 801 / 5730) loss: 1.199963
(Iteration 821 / 5730) loss: 1.015277
(Iteration 841 / 5730) loss: 1.172329
(Iteration 861 / 5730) loss: 1.272491
(Iteration 881 / 5730) loss: 1.035057
(Iteration 901 / 5730) loss: 1.089504
(Iteration 921 / 5730) loss: 1.063471
(Iteration 941 / 5730) loss: 1.006050
(Iteration 961 / 5730) loss: 1.131010
(Iteration 981 / 5730) loss: 1.055226
(Iteration 1001 / 5730) loss: 1.220967
(Iteration 1021 / 5730) loss: 1.035626
(Iteration 1041 / 5730) loss: 1.187960
(Iteration 1061 / 5730) loss: 0.990191
```

```
(Iteration 1081 / 5730) loss: 0.880529
(Iteration 1101 / 5730) loss: 1.076975
(Iteration 1121 / 5730) loss: 1.132610
(Iteration 1141 / 5730) loss: 0.800732
(Epoch 3 / 15) train acc: 0.633000; val_acc: 0.609000
(Iteration 1161 / 5730) loss: 1.090642
(Iteration 1181 / 5730) loss: 1.331991
(Iteration 1201 / 5730) loss: 0.982377
(Iteration 1221 / 5730) loss: 1.047051
(Iteration 1241 / 5730) loss: 1.067707
(Iteration 1261 / 5730) loss: 0.959963
(Iteration 1281 / 5730) loss: 1.047880
(Iteration 1301 / 5730) loss: 0.945537
(Iteration 1321 / 5730) loss: 1.092515
(Iteration 1341 / 5730) loss: 0.921123
(Iteration 1361 / 5730) loss: 0.885517
(Iteration 1381 / 5730) loss: 0.960253
(Iteration 1401 / 5730) loss: 1.023843
(Iteration 1421 / 5730) loss: 1.011569
(Iteration 1441 / 5730) loss: 1.013666
(Iteration 1461 / 5730) loss: 1.334589
(Iteration 1481 / 5730) loss: 1.022286
(Iteration 1501 / 5730) loss: 0.872740
(Iteration 1521 / 5730) loss: 0.937490
(Epoch 4 / 15) train acc: 0.611000; val_acc: 0.595000
(Iteration 1541 / 5730) loss: 1.028597
(Iteration 1561 / 5730) loss: 0.876702
(Iteration 1581 / 5730) loss: 1.132168
(Iteration 1601 / 5730) loss: 0.929874
(Iteration 1621 / 5730) loss: 1.100173
(Iteration 1641 / 5730) loss: 1.092112
(Iteration 1661 / 5730) loss: 1.204253
(Iteration 1681 / 5730) loss: 0.998772
(Iteration 1701 / 5730) loss: 0.812532
(Iteration 1721 / 5730) loss: 1.070772
(Iteration 1741 / 5730) loss: 0.889282
(Iteration 1761 / 5730) loss: 0.962697
(Iteration 1781 / 5730) loss: 0.916171
(Iteration 1801 / 5730) loss: 0.896096
(Iteration 1821 / 5730) loss: 1.000129
(Iteration 1841 / 5730) loss: 0.843282
(Iteration 1861 / 5730) loss: 0.933782
(Iteration 1881 / 5730) loss: 0.877623
(Iteration 1901 / 5730) loss: 0.770611
(Epoch 5 / 15) train acc: 0.687000; val_acc: 0.632000
(Iteration 1921 / 5730) loss: 1.034500
(Iteration 1941 / 5730) loss: 1.072820
(Iteration 1961 / 5730) loss: 0.943722
(Iteration 1981 / 5730) loss: 0.903900
(Iteration 2001 / 5730) loss: 0.978961
(Iteration 2021 / 5730) loss: 0.944058
(Iteration 2041 / 5730) loss: 0.982706
(Iteration 2061 / 5730) loss: 0.945340
(Iteration 2081 / 5730) loss: 0.784325
(Iteration 2101 / 5730) loss: 0.766858
(Iteration 2121 / 5730) loss: 0.926472
(Iteration 2141 / 5730) loss: 0.905241
```

```
(Iteration 2161 / 5730) loss: 0.761714
(Iteration 2181 / 5730) loss: 0.817416
(Iteration 2201 / 5730) loss: 0.950651
(Iteration 2221 / 5730) loss: 0.817195
(Iteration 2241 / 5730) loss: 1.030922
(Iteration 2261 / 5730) loss: 1.037213
(Iteration 2281 / 5730) loss: 0.966074
(Epoch 6 / 15) train acc: 0.679000; val_acc: 0.628000
(Iteration 2301 / 5730) loss: 0.926695
(Iteration 2321 / 5730) loss: 1.200794
(Iteration 2341 / 5730) loss: 0.728167
(Iteration 2361 / 5730) loss: 0.849453
(Iteration 2381 / 5730) loss: 0.850361
(Iteration 2401 / 5730) loss: 0.951400
(Iteration 2421 / 5730) loss: 0.863175
(Iteration 2441 / 5730) loss: 0.938395
(Iteration 2461 / 5730) loss: 0.745604
(Iteration 2481 / 5730) loss: 0.849992
(Iteration 2501 / 5730) loss: 1.047825
(Iteration 2521 / 5730) loss: 0.984919
(Iteration 2541 / 5730) loss: 0.773999
(Iteration 2561 / 5730) loss: 0.876067
(Iteration 2581 / 5730) loss: 0.880775
(Iteration 2601 / 5730) loss: 0.753403
(Iteration 2621 / 5730) loss: 1.063312
(Iteration 2641 / 5730) loss: 0.764027
(Iteration 2661 / 5730) loss: 0.660783
(Epoch 7 / 15) train acc: 0.704000; val_acc: 0.657000
(Iteration 2681 / 5730) loss: 0.875050
(Iteration 2701 / 5730) loss: 0.731897
(Iteration 2721 / 5730) loss: 0.832229
(Iteration 2741 / 5730) loss: 0.827563
(Iteration 2761 / 5730) loss: 1.018010
(Iteration 2781 / 5730) loss: 0.782663
(Iteration 2801 / 5730) loss: 0.953400
(Iteration 2821 / 5730) loss: 0.898229
(Iteration 2841 / 5730) loss: 0.855345
(Iteration 2861 / 5730) loss: 1.111483
(Iteration 2881 / 5730) loss: 0.789999
(Iteration 2901 / 5730) loss: 0.883426
(Iteration 2921 / 5730) loss: 0.902819
(Iteration 2941 / 5730) loss: 0.746762
(Iteration 2961 / 5730) loss: 0.995886
(Iteration 2981 / 5730) loss: 0.811427
(Iteration 3001 / 5730) loss: 0.834081
(Iteration 3021 / 5730) loss: 0.952963
(Iteration 3041 / 5730) loss: 0.944784
(Epoch 8 / 15) train acc: 0.711000; val_acc: 0.634000
(Iteration 3061 / 5730) loss: 0.837555
(Iteration 3081 / 5730) loss: 0.694587
(Iteration 3101 / 5730) loss: 0.821755
(Iteration 3121 / 5730) loss: 0.815190
(Iteration 3141 / 5730) loss: 0.768178
(Iteration 3161 / 5730) loss: 0.750169
(Iteration 3181 / 5730) loss: 0.842079
(Iteration 3201 / 5730) loss: 0.713645
(Iteration 3221 / 5730) loss: 0.853219
```

```
(Iteration 3241 / 5730) loss: 0.749841
(Iteration 3261 / 5730) loss: 0.806953
(Iteration 3281 / 5730) loss: 0.747577
(Iteration 3301 / 5730) loss: 0.759039
(Iteration 3321 / 5730) loss: 0.761904
(Iteration 3341 / 5730) loss: 0.769731
(Iteration 3361 / 5730) loss: 1.103440
(Iteration 3381 / 5730) loss: 0.806305
(Iteration 3401 / 5730) loss: 0.722995
(Iteration 3421 / 5730) loss: 0.910105
(Epoch 9 / 15) train acc: 0.741000; val_acc: 0.640000
(Iteration 3441 / 5730) loss: 0.789341
(Iteration 3461 / 5730) loss: 0.794981
(Iteration 3481 / 5730) loss: 0.808493
(Iteration 3501 / 5730) loss: 0.728320
(Iteration 3521 / 5730) loss: 0.813904
(Iteration 3541 / 5730) loss: 0.811307
(Iteration 3561 / 5730) loss: 0.813526
(Iteration 3581 / 5730) loss: 0.965592
(Iteration 3601 / 5730) loss: 1.043618
(Iteration 3621 / 5730) loss: 0.826015
(Iteration 3641 / 5730) loss: 0.874432
(Iteration 3661 / 5730) loss: 0.640500
(Iteration 3681 / 5730) loss: 0.719851
(Iteration 3701 / 5730) loss: 1.014668
(Iteration 3721 / 5730) loss: 0.896639
(Iteration 3741 / 5730) loss: 0.831060
(Iteration 3761 / 5730) loss: 0.738823
(Iteration 3781 / 5730) loss: 0.921722
(Iteration 3801 / 5730) loss: 0.878072
(Epoch 10 / 15) train acc: 0.692000; val_acc: 0.628000
(Iteration 3821 / 5730) loss: 0.881993
(Iteration 3841 / 5730) loss: 1.087112
(Iteration 3861 / 5730) loss: 0.659757
(Iteration 3881 / 5730) loss: 0.844886
(Iteration 3901 / 5730) loss: 0.838179
(Iteration 3921 / 5730) loss: 0.731724
(Iteration 3941 / 5730) loss: 0.667375
(Iteration 3961 / 5730) loss: 0.922343
(Iteration 3981 / 5730) loss: 0.805053
(Iteration 4001 / 5730) loss: 0.883172
(Iteration 4021 / 5730) loss: 0.736805
(Iteration 4041 / 5730) loss: 0.796258
(Iteration 4061 / 5730) loss: 0.598023
(Iteration 4081 / 5730) loss: 0.855600
(Iteration 4101 / 5730) loss: 0.630917
(Iteration 4121 / 5730) loss: 0.715877
(Iteration 4141 / 5730) loss: 0.509960
(Iteration 4161 / 5730) loss: 0.759190
(Iteration 4181 / 5730) loss: 0.613843
(Iteration 4201 / 5730) loss: 0.645178
(Epoch 11 / 15) train acc: 0.707000; val_acc: 0.601000
(Iteration 4221 / 5730) loss: 0.963549
(Iteration 4241 / 5730) loss: 0.704584
(Iteration 4261 / 5730) loss: 0.701464
(Iteration 4281 / 5730) loss: 0.718884
(Iteration 4301 / 5730) loss: 0.730558
```

```
(Iteration 4321 / 5730) loss: 0.632395
(Iteration 4341 / 5730) loss: 0.612482
(Iteration 4361 / 5730) loss: 0.668974
(Iteration 4381 / 5730) loss: 0.867646
(Iteration 4401 / 5730) loss: 0.687785
(Iteration 4421 / 5730) loss: 0.728129
(Iteration 4441 / 5730) loss: 0.708198
(Iteration 4461 / 5730) loss: 0.762829
(Iteration 4481 / 5730) loss: 0.719877
(Iteration 4501 / 5730) loss: 0.830460
(Iteration 4521 / 5730) loss: 0.923217
(Iteration 4541 / 5730) loss: 0.740081
(Iteration 4561 / 5730) loss: 0.800896
(Iteration 4581 / 5730) loss: 0.660932
(Epoch 12 / 15) train acc: 0.729000; val_acc: 0.655000
(Iteration 4601 / 5730) loss: 0.720061
(Iteration 4621 / 5730) loss: 0.676229
(Iteration 4641 / 5730) loss: 0.880973
(Iteration 4661 / 5730) loss: 0.705091
(Iteration 4681 / 5730) loss: 0.699740
(Iteration 4701 / 5730) loss: 0.609353
(Iteration 4721 / 5730) loss: 0.857134
(Iteration 4741 / 5730) loss: 0.729417
(Iteration 4761 / 5730) loss: 0.763072
(Iteration 4781 / 5730) loss: 0.818449
(Iteration 4801 / 5730) loss: 0.773662
(Iteration 4821 / 5730) loss: 0.895202
(Iteration 4841 / 5730) loss: 0.925682
(Iteration 4861 / 5730) loss: 0.781881
(Iteration 4881 / 5730) loss: 0.700775
(Iteration 4901 / 5730) loss: 0.856194
(Iteration 4921 / 5730) loss: 0.669190
(Iteration 4941 / 5730) loss: 0.847880
(Iteration 4961 / 5730) loss: 0.639630
(Epoch 13 / 15) train acc: 0.750000; val_acc: 0.639000
(Iteration 4981 / 5730) loss: 0.523683
(Iteration 5001 / 5730) loss: 0.764845
(Iteration 5021 / 5730) loss: 0.708426
(Iteration 5041 / 5730) loss: 0.773580
(Iteration 5061 / 5730) loss: 0.661314
(Iteration 5081 / 5730) loss: 0.582456
(Iteration 5101 / 5730) loss: 0.738108
(Iteration 5121 / 5730) loss: 0.901258
(Iteration 5141 / 5730) loss: 0.705411
(Iteration 5161 / 5730) loss: 0.678164
(Iteration 5181 / 5730) loss: 0.735594
(Iteration 5201 / 5730) loss: 0.920688
(Iteration 5221 / 5730) loss: 0.943875
(Iteration 5241 / 5730) loss: 0.883143
(Iteration 5261 / 5730) loss: 0.784406
(Iteration 5281 / 5730) loss: 0.787113
(Iteration 5301 / 5730) loss: 0.989408
(Iteration 5321 / 5730) loss: 0.717533
(Iteration 5341 / 5730) loss: 0.803206
(Epoch 14 / 15) train acc: 0.757000; val_acc: 0.647000
(Iteration 5361 / 5730) loss: 0.804627
(Iteration 5381 / 5730) loss: 0.674884
```

```
(Iteration 5401 / 5730) loss: 0.760615
(Iteration 5421 / 5730) loss: 0.815537
(Iteration 5441 / 5730) loss: 0.734104
(Iteration 5461 / 5730) loss: 0.764333
(Iteration 5481 / 5730) loss: 0.823910
(Iteration 5501 / 5730) loss: 0.800798
(Iteration 5521 / 5730) loss: 0.613245
(Iteration 5541 / 5730) loss: 0.620098
(Iteration 5561 / 5730) loss: 0.894341
(Iteration 5581 / 5730) loss: 0.603590
(Iteration 5601 / 5730) loss: 0.830513
(Iteration 5621 / 5730) loss: 0.872919
(Iteration 5641 / 5730) loss: 0.755896
(Iteration 5661 / 5730) loss: 0.766207
(Iteration 5681 / 5730) loss: 0.720361
(Iteration 5701 / 5730) loss: 0.923137
(Iteration 5721 / 5730) loss: 0.759570
(Epoch 15 / 15) train acc: 0.771000; val_acc: 0.669000
```