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.

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]:

```python
# As usual, a bit of setup

import numpy as np
import matplotlib.pyplot as plt
from nndl.cnn import *
from utils.data_utils import get_CIFAR10_data
from utils.gradient_check import eval_numerical_gradient_array, eval_numerical_gradient
from nndl.layers import *
from nndl.conv_layers import *
from utils.fast_layers import *
from utils.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))))
```

In [2]:

```python
# 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_train: (49000,)
X_val: (1000, 3, 32, 32)
y_val: (1000,)
X_test: (1000, 3, 32, 32)
y_test: (1000,)
```

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 [4]:

```
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_name], verbose=False,
    e = rel_error(param_grad_num, grads[param_name])
    print('{} max relative error: {}'.format(param_name, rel_error(param_grad_num, grads
```

```
W1 max relative error: 0.002268986592757279
W2 max relative error: 0.0021176341224491218
W3 max relative error: 5.226928034487663e-06
b1 max relative error: 4.196696704863285e-06
b2 max relative error: 1.487895884269271e-07
b3 max relative error: 1.4628540331904963e-09
```

Overfit small dataset

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

In [5]:

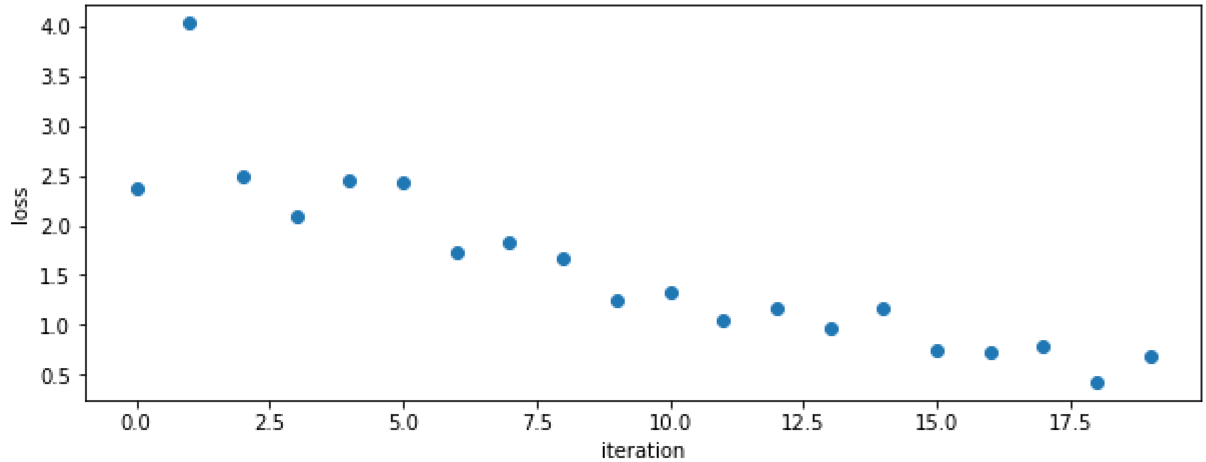
```
num_train = 100
small_data = {
  'X_train': data['X_train'][:num_train],
  'y_train': data['y_train'][:num_train],
```

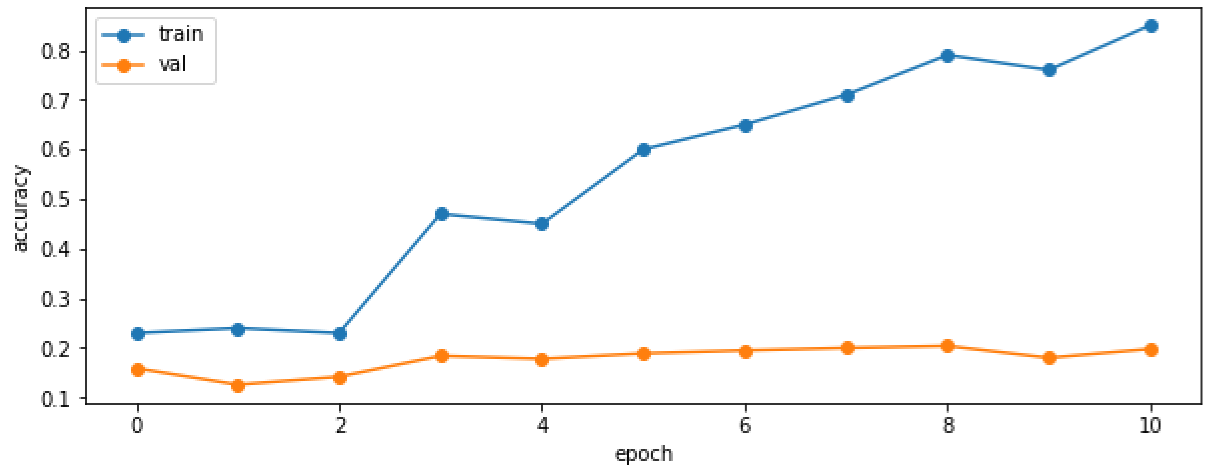
```python
  'X_val': data['X_val'],
  'y_val': data['y_val'],
}

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: 2.380656
(Epoch 0 / 10) train acc: 0.230000; val_acc: 0.159000
(Iteration 2 / 20) loss: 4.038913
(Epoch 1 / 10) train acc: 0.240000; val_acc: 0.126000
(Iteration 3 / 20) loss: 2.483374
(Iteration 4 / 20) loss: 2.097788
(Epoch 2 / 10) train acc: 0.230000; val_acc: 0.142000
(Iteration 5 / 20) loss: 2.446642
(Iteration 6 / 20) loss: 2.437417
(Epoch 3 / 10) train acc: 0.470000; val_acc: 0.184000
(Iteration 7 / 20) loss: 1.723952
(Iteration 8 / 20) loss: 1.821720
(Epoch 4 / 10) train acc: 0.450000; val_acc: 0.178000
(Iteration 9 / 20) loss: 1.672315
(Iteration 10 / 20) loss: 1.241660
(Epoch 5 / 10) train acc: 0.600000; val_acc: 0.189000
(Iteration 11 / 20) loss: 1.335591
(Iteration 12 / 20) loss: 1.040599
(Epoch 6 / 10) train acc: 0.650000; val_acc: 0.195000
(Iteration 13 / 20) loss: 1.166686
(Iteration 14 / 20) loss: 0.972074
(Epoch 7 / 10) train acc: 0.710000; val_acc: 0.200000
(Iteration 15 / 20) loss: 1.166310
(Iteration 16 / 20) loss: 0.747452
(Epoch 8 / 10) train acc: 0.790000; val_acc: 0.204000
(Iteration 17 / 20) loss: 0.725688
(Iteration 18 / 20) loss: 0.794187
(Epoch 9 / 10) train acc: 0.760000; val_acc: 0.180000
(Iteration 19 / 20) loss: 0.433434
(Iteration 20 / 20) loss: 0.691781
(Epoch 10 / 10) train acc: 0.850000; val_acc: 0.198000
```

In [6]:

```python
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 [7]:

```
model = ThreeLayerConvNet(weight_scale=0.001, hidden_dim=500, reg=0.001)

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: 2.304783
(Epoch 0 / 1) train acc: 0.102000; val_acc: 0.113000
(Iteration 21 / 980) loss: 2.074944
(Iteration 41 / 980) loss: 2.332429
(Iteration 61 / 980) loss: 1.825277
(Iteration 81 / 980) loss: 1.835692
(Iteration 101 / 980) loss: 1.916303
(Iteration 121 / 980) loss: 1.621939
(Iteration 141 / 980) loss: 1.465717
(Iteration 161 / 980) loss: 1.819977
(Iteration 181 / 980) loss: 1.743110
(Iteration 201 / 980) loss: 1.670612
(Iteration 221 / 980) loss: 1.865526
(Iteration 241 / 980) loss: 1.698929
(Iteration 261 / 980) loss: 1.720045
(Iteration 281 / 980) loss: 1.807330
(Iteration 301 / 980) loss: 1.909947
(Iteration 321 / 980) loss: 1.593170
(Iteration 341 / 980) loss: 1.849560
(Iteration 361 / 980) loss: 1.610693
(Iteration 381 / 980) loss: 1.802914
(Iteration 401 / 980) loss: 1.591736
(Iteration 421 / 980) loss: 1.561139
(Iteration 441 / 980) loss: 1.604666
(Iteration 461 / 980) loss: 1.673824
(Iteration 481 / 980) loss: 1.613188
(Iteration 501 / 980) loss: 1.643375
(Iteration 521 / 980) loss: 1.579830
(Iteration 541 / 980) loss: 1.431830
(Iteration 561 / 980) loss: 1.558496
(Iteration 581 / 980) loss: 1.815331
(Iteration 601 / 980) loss: 1.741974
(Iteration 621 / 980) loss: 1.871260
(Iteration 641 / 980) loss: 1.741922
(Iteration 661 / 980) loss: 1.948502
(Iteration 681 / 980) loss: 1.667782
(Iteration 701 / 980) loss: 1.505128
(Iteration 721 / 980) loss: 1.511095
(Iteration 741 / 980) loss: 2.047607
(Iteration 761 / 980) loss: 1.614136
(Iteration 781 / 980) loss: 1.481369
(Iteration 801 / 980) loss: 1.472241
(Iteration 821 / 980) loss: 1.406312
(Iteration 841 / 980) loss: 1.565292
(Iteration 861 / 980) loss: 1.417483
(Iteration 881 / 980) loss: 1.277716
(Iteration 901 / 980) loss: 1.976424
(Iteration 921 / 980) loss: 1.820521
(Iteration 941 / 980) loss: 1.568014
(Iteration 961 / 980) loss: 1.555147
(Epoch 1 / 1) train acc: 0.472000; val_acc: 0.480000
```

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 aafter 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 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 [10]:

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

model = ThreeLayerConvNet(weight_scale = 0.001, hidden_dim = 500, reg = 0.001,
                          filter_size = 3, num_filters = 64)

solver = Solver(model, data,
                num_epochs = 50, batch_size = 500,
                update_rule = 'adam',
                optim_config = {
                  'learning_rate': 1e-3,
                },
                lr_decay = 0.9,
                verbose = True, print_every = 20)
solver.train()

y_test_pred = np.argmax(model.loss(data['X_test']), axis = 1)
y_val_pred = np.argmax(model.loss(data['X_val']), axis = 1)
print("\nTest set accuracy: {}".format(np.mean(np.equal(y_test_pred, data['y_test']))))
print("Validation set accuracy: {}".format(np.mean(np.equal(y_val_pred, data['y_val']))))

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

```
(Iteration 1 / 4900) loss: 2.306748
(Epoch 0 / 50) train acc: 0.125000; val_acc: 0.108000
(Iteration 21 / 4900) loss: 1.849256
(Iteration 41 / 4900) loss: 1.524093
(Iteration 61 / 4900) loss: 1.368649
(Iteration 81 / 4900) loss: 1.434639
(Epoch 1 / 50) train acc: 0.555000; val_acc: 0.536000
(Iteration 101 / 4900) loss: 1.320356
(Iteration 121 / 4900) loss: 1.279837
(Iteration 141 / 4900) loss: 1.222214
(Iteration 161 / 4900) loss: 1.155338
(Iteration 181 / 4900) loss: 1.274913
(Epoch 2 / 50) train acc: 0.602000; val_acc: 0.590000
(Iteration 201 / 4900) loss: 1.140279
(Iteration 221 / 4900) loss: 1.181593
(Iteration 241 / 4900) loss: 1.116856
(Iteration 261 / 4900) loss: 1.090370
(Iteration 281 / 4900) loss: 1.019950
(Epoch 3 / 50) train acc: 0.648000; val_acc: 0.632000
(Iteration 301 / 4900) loss: 1.011179
(Iteration 321 / 4900) loss: 1.082931
(Iteration 341 / 4900) loss: 1.021899
(Iteration 361 / 4900) loss: 0.899474
(Iteration 381 / 4900) loss: 0.964524
(Epoch 4 / 50) train acc: 0.722000; val_acc: 0.636000
(Iteration 401 / 4900) loss: 0.874312
(Iteration 421 / 4900) loss: 0.863526
(Iteration 441 / 4900) loss: 0.806440
(Iteration 461 / 4900) loss: 0.794834
(Iteration 481 / 4900) loss: 0.791156
(Epoch 5 / 50) train acc: 0.749000; val_acc: 0.652000
(Iteration 501 / 4900) loss: 0.843182
(Iteration 521 / 4900) loss: 0.825679
(Iteration 541 / 4900) loss: 0.798971
(Iteration 561 / 4900) loss: 0.780464
(Iteration 581 / 4900) loss: 0.751151
(Epoch 6 / 50) train acc: 0.796000; val_acc: 0.647000
(Iteration 601 / 4900) loss: 0.702370
(Iteration 621 / 4900) loss: 0.677453
(Iteration 641 / 4900) loss: 0.762879
(Iteration 661 / 4900) loss: 0.701149
(Iteration 681 / 4900) loss: 0.656555
(Epoch 7 / 50) train acc: 0.838000; val_acc: 0.656000
(Iteration 701 / 4900) loss: 0.602177
(Iteration 721 / 4900) loss: 0.601657
(Iteration 741 / 4900) loss: 0.535341
(Iteration 761 / 4900) loss: 0.581571
(Iteration 781 / 4900) loss: 0.614937
(Epoch 8 / 50) train acc: 0.848000; val_acc: 0.667000
(Iteration 801 / 4900) loss: 0.545222
(Iteration 821 / 4900) loss: 0.566455
(Iteration 841 / 4900) loss: 0.542922
(Iteration 861 / 4900) loss: 0.553123
(Iteration 881 / 4900) loss: 0.564563
(Epoch 9 / 50) train acc: 0.845000; val_acc: 0.663000
(Iteration 901 / 4900) loss: 0.535859
(Iteration 921 / 4900) loss: 0.565521
(Iteration 941 / 4900) loss: 0.502146
(Iteration 961 / 4900) loss: 0.492933
(Epoch 10 / 50) train acc: 0.878000; val_acc: 0.672000
(Iteration 981 / 4900) loss: 0.507488
(Iteration 1001 / 4900) loss: 0.472753
(Iteration 1021 / 4900) loss: 0.405617
(Iteration 1041 / 4900) loss: 0.457744
(Iteration 1061 / 4900) loss: 0.482305
(Epoch 11 / 50) train acc: 0.922000; val_acc: 0.681000
```

```
(Iteration 1081 / 4900) loss: 0.410904
(Iteration 1101 / 4900) loss: 0.392253
(Iteration 1121 / 4900) loss: 0.441741
(Iteration 1141 / 4900) loss: 0.375415
(Iteration 1161 / 4900) loss: 0.359357
(Epoch 12 / 50) train acc: 0.935000; val_acc: 0.683000
(Iteration 1181 / 4900) loss: 0.375785
(Iteration 1201 / 4900) loss: 0.360305
(Iteration 1221 / 4900) loss: 0.333513
(Iteration 1241 / 4900) loss: 0.355085
(Iteration 1261 / 4900) loss: 0.320918
(Epoch 13 / 50) train acc: 0.939000; val_acc: 0.675000
(Iteration 1281 / 4900) loss: 0.325334
(Iteration 1301 / 4900) loss: 0.323204
(Iteration 1321 / 4900) loss: 0.313124
(Iteration 1341 / 4900) loss: 0.301074
(Iteration 1361 / 4900) loss: 0.322299
(Epoch 14 / 50) train acc: 0.946000; val_acc: 0.666000
(Iteration 1381 / 4900) loss: 0.268913
(Iteration 1401 / 4900) loss: 0.295817
(Iteration 1421 / 4900) loss: 0.280274
(Iteration 1441 / 4900) loss: 0.283640
(Iteration 1461 / 4900) loss: 0.274886
(Epoch 15 / 50) train acc: 0.960000; val_acc: 0.695000
(Iteration 1481 / 4900) loss: 0.248470
(Iteration 1501 / 4900) loss: 0.239041
(Iteration 1521 / 4900) loss: 0.247193
(Iteration 1541 / 4900) loss: 0.241076
(Iteration 1561 / 4900) loss: 0.264690
(Epoch 16 / 50) train acc: 0.966000; val_acc: 0.684000
(Iteration 1581 / 4900) loss: 0.241561
(Iteration 1601 / 4900) loss: 0.231545
(Iteration 1621 / 4900) loss: 0.248332
(Iteration 1641 / 4900) loss: 0.221348
(Iteration 1661 / 4900) loss: 0.233136
(Epoch 17 / 50) train acc: 0.991000; val_acc: 0.684000
(Iteration 1681 / 4900) loss: 0.224402
(Iteration 1701 / 4900) loss: 0.203306
(Iteration 1721 / 4900) loss: 0.210451
(Iteration 1741 / 4900) loss: 0.200417
(Iteration 1761 / 4900) loss: 0.215092
(Epoch 18 / 50) train acc: 0.993000; val_acc: 0.688000
(Iteration 1781 / 4900) loss: 0.194550
(Iteration 1801 / 4900) loss: 0.194386
(Iteration 1821 / 4900) loss: 0.180452
(Iteration 1841 / 4900) loss: 0.191180
(Iteration 1861 / 4900) loss: 0.184778
(Epoch 19 / 50) train acc: 0.997000; val_acc: 0.690000
(Iteration 1881 / 4900) loss: 0.185082
(Iteration 1901 / 4900) loss: 0.192765
(Iteration 1921 / 4900) loss: 0.190556
(Iteration 1941 / 4900) loss: 0.183072
(Epoch 20 / 50) train acc: 0.997000; val_acc: 0.691000
(Iteration 1961 / 4900) loss: 0.173359
(Iteration 1981 / 4900) loss: 0.182031
(Iteration 2001 / 4900) loss: 0.174280
(Iteration 2021 / 4900) loss: 0.172336
(Iteration 2041 / 4900) loss: 0.179485
(Epoch 21 / 50) train acc: 0.997000; val_acc: 0.681000
(Iteration 2061 / 4900) loss: 0.170681
(Iteration 2081 / 4900) loss: 0.166374
(Iteration 2101 / 4900) loss: 0.172466
(Iteration 2121 / 4900) loss: 0.160887
(Iteration 2141 / 4900) loss: 0.156633
(Epoch 22 / 50) train acc: 0.999000; val_acc: 0.680000
(Iteration 2161 / 4900) loss: 0.162382
```

```
(Iteration 2181 / 4900) loss: 0.155620
(Iteration 2201 / 4900) loss: 0.161025
(Iteration 2221 / 4900) loss: 0.157615
(Iteration 2241 / 4900) loss: 0.153030
(Epoch 23 / 50) train acc: 0.995000; val_acc: 0.682000
(Iteration 2261 / 4900) loss: 0.151908
(Iteration 2281 / 4900) loss: 0.152797
(Iteration 2301 / 4900) loss: 0.147538
(Iteration 2321 / 4900) loss: 0.161522
(Iteration 2341 / 4900) loss: 0.149376
(Epoch 24 / 50) train acc: 0.997000; val_acc: 0.683000
(Iteration 2361 / 4900) loss: 0.148321
(Iteration 2381 / 4900) loss: 0.151042
(Iteration 2401 / 4900) loss: 0.144019
(Iteration 2421 / 4900) loss: 0.145643
(Iteration 2441 / 4900) loss: 0.149745
(Epoch 25 / 50) train acc: 0.999000; val_acc: 0.687000
(Iteration 2461 / 4900) loss: 0.145662
(Iteration 2481 / 4900) loss: 0.147911
(Iteration 2501 / 4900) loss: 0.146025
(Iteration 2521 / 4900) loss: 0.145308
(Iteration 2541 / 4900) loss: 0.140447
(Epoch 26 / 50) train acc: 1.000000; val_acc: 0.687000
(Iteration 2561 / 4900) loss: 0.141427
(Iteration 2581 / 4900) loss: 0.138445
(Iteration 2601 / 4900) loss: 0.141254
(Iteration 2621 / 4900) loss: 0.138364
(Iteration 2641 / 4900) loss: 0.138283
(Epoch 27 / 50) train acc: 0.999000; val_acc: 0.693000
(Iteration 2661 / 4900) loss: 0.137330
(Iteration 2681 / 4900) loss: 0.138596
(Iteration 2701 / 4900) loss: 0.137016
(Iteration 2721 / 4900) loss: 0.138736
(Iteration 2741 / 4900) loss: 0.133788
(Epoch 28 / 50) train acc: 0.998000; val_acc: 0.681000
(Iteration 2761 / 4900) loss: 0.140178
(Iteration 2781 / 4900) loss: 0.133195
(Iteration 2801 / 4900) loss: 0.134731
(Iteration 2821 / 4900) loss: 0.132483
(Iteration 2841 / 4900) loss: 0.134764
(Epoch 29 / 50) train acc: 1.000000; val_acc: 0.692000
(Iteration 2861 / 4900) loss: 0.140545
(Iteration 2881 / 4900) loss: 0.131893
(Iteration 2901 / 4900) loss: 0.133738
(Iteration 2921 / 4900) loss: 0.133498
(Epoch 30 / 50) train acc: 1.000000; val_acc: 0.686000
(Iteration 2941 / 4900) loss: 0.132696
(Iteration 2961 / 4900) loss: 0.134023
(Iteration 2981 / 4900) loss: 0.131690
(Iteration 3001 / 4900) loss: 0.128803
(Iteration 3021 / 4900) loss: 0.125610
(Epoch 31 / 50) train acc: 1.000000; val_acc: 0.689000
(Iteration 3041 / 4900) loss: 0.128125
(Iteration 3061 / 4900) loss: 0.130076
(Iteration 3081 / 4900) loss: 0.129865
(Iteration 3101 / 4900) loss: 0.127056
(Iteration 3121 / 4900) loss: 0.128843
(Epoch 32 / 50) train acc: 1.000000; val_acc: 0.682000
(Iteration 3141 / 4900) loss: 0.128974
(Iteration 3161 / 4900) loss: 0.129386
(Iteration 3181 / 4900) loss: 0.128048
(Iteration 3201 / 4900) loss: 0.123976
(Iteration 3221 / 4900) loss: 0.125482
(Epoch 33 / 50) train acc: 1.000000; val_acc: 0.682000
(Iteration 3241 / 4900) loss: 0.126683
(Iteration 3261 / 4900) loss: 0.124154
```

```
(Iteration 3281 / 4900) loss: 0.121943
(Iteration 3301 / 4900) loss: 0.123917
(Iteration 3321 / 4900) loss: 0.123658
(Epoch 34 / 50) train acc: 0.999000; val_acc: 0.685000
(Iteration 3341 / 4900) loss: 0.125339
(Iteration 3361 / 4900) loss: 0.123392
(Iteration 3381 / 4900) loss: 0.123290
(Iteration 3401 / 4900) loss: 0.122007
(Iteration 3421 / 4900) loss: 0.124383
(Epoch 35 / 50) train acc: 1.000000; val_acc: 0.685000
(Iteration 3441 / 4900) loss: 0.123018
(Iteration 3461 / 4900) loss: 0.121734
(Iteration 3481 / 4900) loss: 0.122810
(Iteration 3501 / 4900) loss: 0.123955
(Iteration 3521 / 4900) loss: 0.122147
(Epoch 36 / 50) train acc: 1.000000; val_acc: 0.686000
(Iteration 3541 / 4900) loss: 0.119142
(Iteration 3561 / 4900) loss: 0.119825
(Iteration 3581 / 4900) loss: 0.122574
(Iteration 3601 / 4900) loss: 0.120766
(Iteration 3621 / 4900) loss: 0.120571
(Epoch 37 / 50) train acc: 1.000000; val_acc: 0.688000
(Iteration 3641 / 4900) loss: 0.119978
(Iteration 3661 / 4900) loss: 0.119570
(Iteration 3681 / 4900) loss: 0.121297
(Iteration 3701 / 4900) loss: 0.120778
(Iteration 3721 / 4900) loss: 0.118083
(Epoch 38 / 50) train acc: 1.000000; val_acc: 0.690000
(Iteration 3741 / 4900) loss: 0.119051
(Iteration 3761 / 4900) loss: 0.120059
(Iteration 3781 / 4900) loss: 0.116883
(Iteration 3801 / 4900) loss: 0.117397
(Iteration 3821 / 4900) loss: 0.120239
(Epoch 39 / 50) train acc: 1.000000; val_acc: 0.688000
(Iteration 3841 / 4900) loss: 0.115992
(Iteration 3861 / 4900) loss: 0.116480
(Iteration 3881 / 4900) loss: 0.118696
(Iteration 3901 / 4900) loss: 0.119531
(Epoch 40 / 50) train acc: 1.000000; val_acc: 0.690000
(Iteration 3921 / 4900) loss: 0.118469
(Iteration 3941 / 4900) loss: 0.118598
(Iteration 3961 / 4900) loss: 0.116193
(Iteration 3981 / 4900) loss: 0.119277
(Iteration 4001 / 4900) loss: 0.118941
(Epoch 41 / 50) train acc: 1.000000; val_acc: 0.690000
(Iteration 4021 / 4900) loss: 0.115371
(Iteration 4041 / 4900) loss: 0.117648
(Iteration 4061 / 4900) loss: 0.115878
(Iteration 4081 / 4900) loss: 0.117066
(Iteration 4101 / 4900) loss: 0.116683
(Epoch 42 / 50) train acc: 0.999000; val_acc: 0.689000
(Iteration 4121 / 4900) loss: 0.116432
(Iteration 4141 / 4900) loss: 0.115464
(Iteration 4161 / 4900) loss: 0.117087
(Iteration 4181 / 4900) loss: 0.114474
(Iteration 4201 / 4900) loss: 0.113973
(Epoch 43 / 50) train acc: 1.000000; val_acc: 0.690000
(Iteration 4221 / 4900) loss: 0.114399
(Iteration 4241 / 4900) loss: 0.116668
(Iteration 4261 / 4900) loss: 0.117323
(Iteration 4281 / 4900) loss: 0.123711
(Iteration 4301 / 4900) loss: 0.117889
(Epoch 44 / 50) train acc: 1.000000; val_acc: 0.692000
(Iteration 4321 / 4900) loss: 0.115961
(Iteration 4341 / 4900) loss: 0.114820
(Iteration 4361 / 4900) loss: 0.116927
```

```
(Iteration 4381 / 4900) loss: 0.113153
(Iteration 4401 / 4900) loss: 0.112966
(Epoch 45 / 50) train acc: 1.000000; val_acc: 0.694000
(Iteration 4421 / 4900) loss: 0.115321
(Iteration 4441 / 4900) loss: 0.115764
(Iteration 4461 / 4900) loss: 0.113756
(Iteration 4481 / 4900) loss: 0.115084
(Iteration 4501 / 4900) loss: 0.116188
(Epoch 46 / 50) train acc: 1.000000; val_acc: 0.688000
(Iteration 4521 / 4900) loss: 0.114638
(Iteration 4541 / 4900) loss: 0.113801
(Iteration 4561 / 4900) loss: 0.112388
(Iteration 4581 / 4900) loss: 0.116278
(Iteration 4601 / 4900) loss: 0.112789
(Epoch 47 / 50) train acc: 1.000000; val_acc: 0.688000
(Iteration 4621 / 4900) loss: 0.113292
(Iteration 4641 / 4900) loss: 0.116315
(Iteration 4661 / 4900) loss: 0.113635
(Iteration 4681 / 4900) loss: 0.113629
(Iteration 4701 / 4900) loss: 0.115671
(Epoch 48 / 50) train acc: 1.000000; val_acc: 0.694000
(Iteration 4721 / 4900) loss: 0.111993
(Iteration 4741 / 4900) loss: 0.112073
(Iteration 4761 / 4900) loss: 0.114578
(Iteration 4781 / 4900) loss: 0.113465
(Iteration 4801 / 4900) loss: 0.122049
(Epoch 49 / 50) train acc: 0.999000; val_acc: 0.690000
(Iteration 4821 / 4900) loss: 0.111105
(Iteration 4841 / 4900) loss: 0.112738
(Iteration 4861 / 4900) loss: 0.112168
(Iteration 4881 / 4900) loss: 0.111772
(Epoch 50 / 50) train acc: 1.000000; val_acc: 0.690000

Test set accuracy: 0.671
Validation set accuracy: 0.695
```