In [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. 'cs231n/assignments/assignment1/'
FOLDERNAME = 'Colab Notebooks/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/MyW Drive/$FOLDERNAME/cs231n/datasets/
!bash get_datasets.sh
%cd /content/drive/MyW Drive/$FOLDERNAME
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mo
unt("/content/drive", force_remount=True).
/content/drive/My Drive/Colab Notebooks/assignment2/cs231n/datasets
/content/drive/My Drive/Colab Notebooks/assignment2
```

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 `cs231n/classifiers/fc_net.py`.

Implement the network initialization, forward pass, and backward pass. Throughout this assignment, you will be implementing layers in `cs231n/layers.py`. You can re-use your implementations for `affine_forward`, `affine_backward`, `relu_forward`, `relu_backward`, and `softmax_loss` from Assignment 1. For right now, don't worry about implementing dropout or batch/layer normalization yet, as you will add those features later.

In [3]:

```
# Setup cell.
import time
import numpy as np
import matplotlib.pyplot as plt
from cs231n.classifiers.fc_net import *
from cs231n.data_utils import get_CIFAR10_data
from cs231n.gradient_check import eval_numerical_gradient, eval_numerical_gradient_a
from cs231n.solver import Solver

%matplotlib inline
plt.rcParams["figure.figsize"] = (10.0, 8.0)  # Set default size of plots.
plt.rcParams["image.interpolation"] = "nearest"
plt.rcParams["image.cmap"] = "gray"

%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 [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,)
```

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.

In [5]:
```
np.random.seed(231)
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
        print(f"{name} relative error: {rel_error(grad_num, grads[name])}")
```

```
Running check with reg =  0
Initial loss:  2.300479089768492
W1 relative error: 1.0252674471656573e-07
W2 relative error: 2.2120479295080622e-05
W3 relative error: 4.5623278736665505e-07
b1 relative error: 4.6600944653202505e-09
b2 relative error: 2.085654276112763e-09
b3 relative error: 1.689724888469736e-10
Running check with reg =  3.14
Initial loss:  7.052114776533016
W1 relative error: 3.904541941902138e-09
W2 relative error: 6.86942277940646e-08
W3 relative error: 3.483989247437803e-08
```

```
b1 relative error: 1.4752427965311745e-08
b2 relative error: 1.4615869332918208e-09
b3 relative error: 1.3200479211447775e-10
```

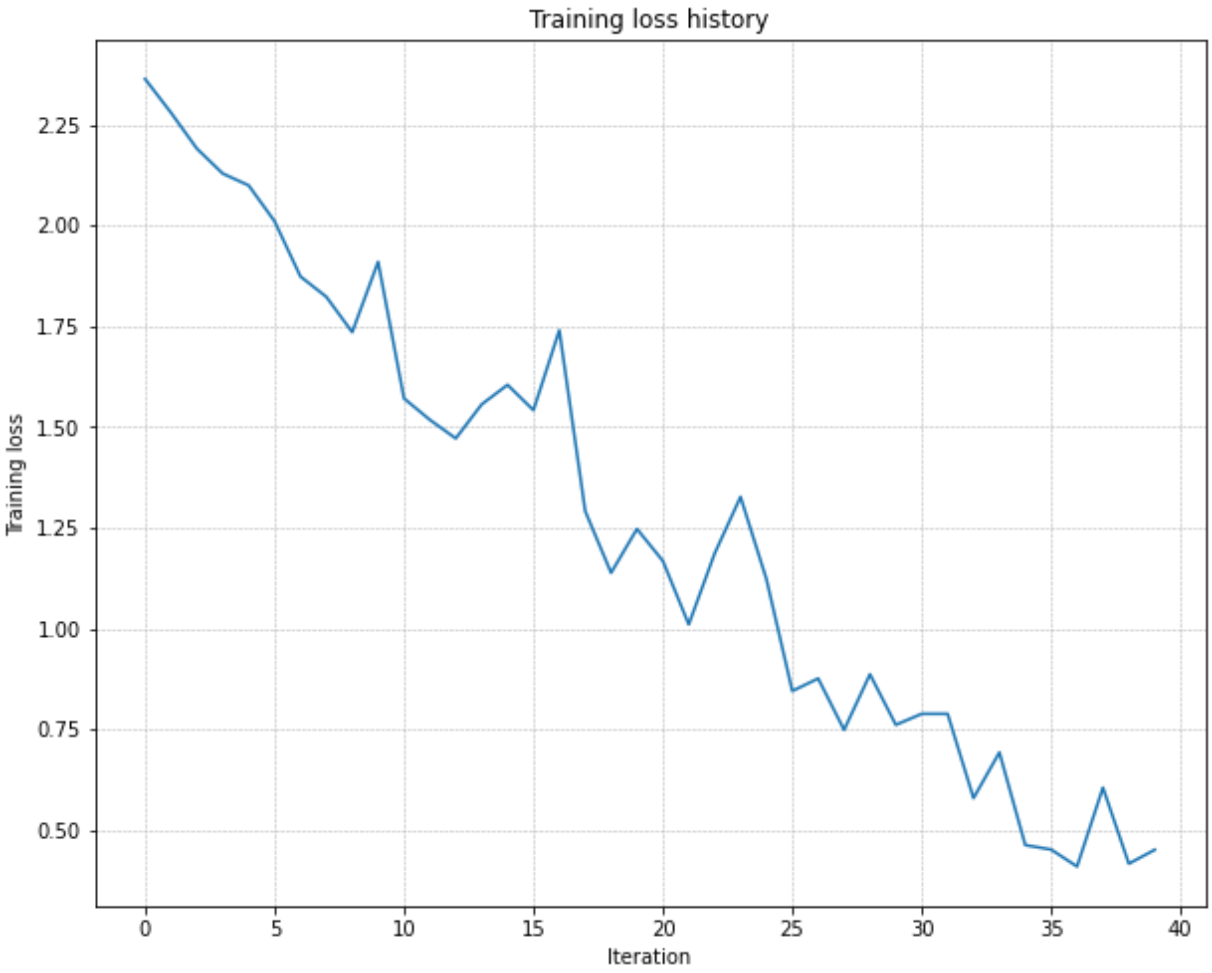
As another sanity check, make sure your network can overfit on a small dataset of 50 images. First, we will try a three-layer network with 100 units in each hidden layer. In the following cell, tweak the **learning rate** and **weight initialization scale** to overfit and achieve 100% training accuracy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

In [6]:

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

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

weight_scale = 1e-2   # Experiment with this!
learning_rate = 4e-3  # Experiment with this!
model = FullyConnectedNet(
    [100, 100],
    weight_scale=weight_scale,
    dtype=np.float64
)
solver = Solver(
    model,
    small_data,
    print_every=10,
    num_epochs=20,
    batch_size=25,
    update_rule="sgd",
    optim_config={"learning_rate": learning_rate},
)
solver.train()

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

```
(Iteration 1 / 40) loss: 2.363364
(Epoch 0 / 20) train acc: 0.080000; val_acc: 0.102000
(Epoch 1 / 20) train acc: 0.160000; val_acc: 0.114000
(Epoch 2 / 20) train acc: 0.320000; val_acc: 0.131000
(Epoch 3 / 20) train acc: 0.360000; val_acc: 0.147000
(Epoch 4 / 20) train acc: 0.380000; val_acc: 0.152000
(Epoch 5 / 20) train acc: 0.380000; val_acc: 0.149000
(Iteration 11 / 40) loss: 1.570807
(Epoch 6 / 20) train acc: 0.460000; val_acc: 0.172000
(Epoch 7 / 20) train acc: 0.500000; val_acc: 0.175000
(Epoch 8 / 20) train acc: 0.600000; val_acc: 0.173000
(Epoch 9 / 20) train acc: 0.680000; val_acc: 0.192000
(Epoch 10 / 20) train acc: 0.760000; val_acc: 0.188000
(Iteration 21 / 40) loss: 1.167700
(Epoch 11 / 20) train acc: 0.820000; val_acc: 0.204000
(Epoch 12 / 20) train acc: 0.820000; val_acc: 0.202000
(Epoch 13 / 20) train acc: 0.780000; val_acc: 0.188000
(Epoch 14 / 20) train acc: 0.780000; val_acc: 0.180000
(Epoch 15 / 20) train acc: 0.920000; val_acc: 0.207000
(Iteration 31 / 40) loss: 0.788213
```

```
(Epoch 16 / 20) train acc: 0.940000; val_acc: 0.206000
(Epoch 17 / 20) train acc: 0.980000; val_acc: 0.210000
(Epoch 18 / 20) train acc: 0.880000; val_acc: 0.202000
(Epoch 19 / 20) train acc: 0.920000; val_acc: 0.180000
(Epoch 20 / 20) train acc: 0.940000; val_acc: 0.200000
```



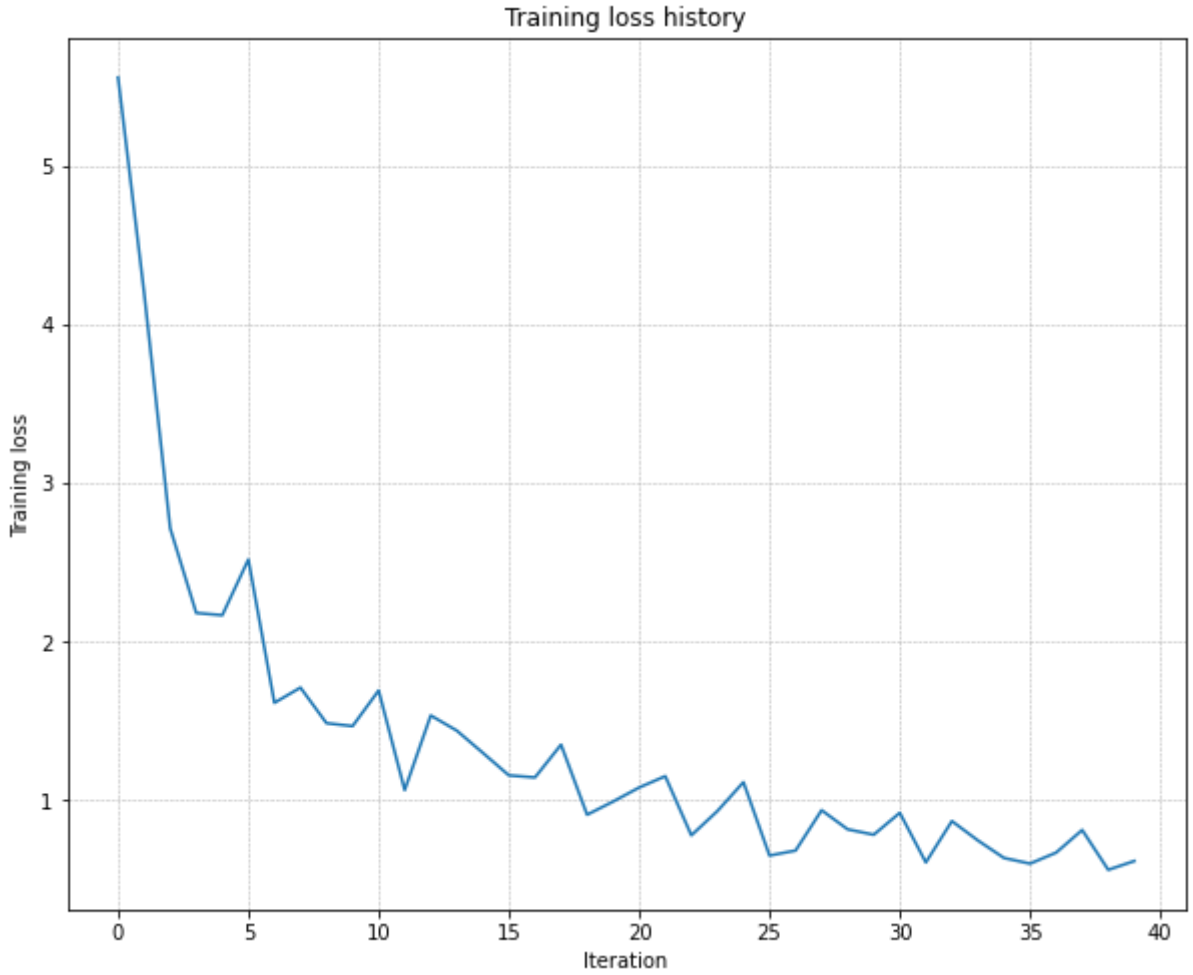
Now, try to use a five-layer network with 100 units on each layer to overfit on 50 training examples. Again, you will have to adjust the learning rate and weight initialization scale, but you should be able to achieve 100% training accuracy within 20 epochs.

In [7]:

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

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

learning_rate = 1e-3 # Experiment with this!
weight_scale = 5e-2  # Experiment with this!
model = FullyConnectedNet(
    [100, 100, 100, 100],
    weight_scale=weight_scale,
    dtype=np.float64
)
solver = Solver(
    model,
    small_data,
    print_every=10,
    num_epochs=20,
```

```
    batch_size=25,
    update_rule='sgd',
    optim_config={'learning_rate': learning_rate},
)
solver.train()

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

```
(Iteration 1 / 40) loss: 5.557032
(Epoch 0 / 20) train acc: 0.160000; val_acc: 0.124000
(Epoch 1 / 20) train acc: 0.140000; val_acc: 0.120000
(Epoch 2 / 20) train acc: 0.360000; val_acc: 0.105000
(Epoch 3 / 20) train acc: 0.400000; val_acc: 0.121000
(Epoch 4 / 20) train acc: 0.440000; val_acc: 0.122000
(Epoch 5 / 20) train acc: 0.540000; val_acc: 0.107000
(Iteration 11 / 40) loss: 1.693275
(Epoch 6 / 20) train acc: 0.580000; val_acc: 0.127000
(Epoch 7 / 20) train acc: 0.660000; val_acc: 0.130000
(Epoch 8 / 20) train acc: 0.680000; val_acc: 0.126000
(Epoch 9 / 20) train acc: 0.700000; val_acc: 0.132000
(Epoch 10 / 20) train acc: 0.720000; val_acc: 0.138000
(Iteration 21 / 40) loss: 1.081837
(Epoch 11 / 20) train acc: 0.760000; val_acc: 0.136000
(Epoch 12 / 20) train acc: 0.820000; val_acc: 0.140000
(Epoch 13 / 20) train acc: 0.820000; val_acc: 0.140000
(Epoch 14 / 20) train acc: 0.800000; val_acc: 0.135000
(Epoch 15 / 20) train acc: 0.840000; val_acc: 0.139000
(Iteration 31 / 40) loss: 0.922168
(Epoch 16 / 20) train acc: 0.860000; val_acc: 0.134000
(Epoch 17 / 20) train acc: 0.860000; val_acc: 0.135000
(Epoch 18 / 20) train acc: 0.900000; val_acc: 0.156000
(Epoch 19 / 20) train acc: 0.900000; val_acc: 0.141000
(Epoch 20 / 20) train acc: 0.920000; val_acc: 0.140000
```



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?

Answer:

레이어가 5개일 때 accuracy를 높이기가 더 힘들었다. 레이어가 5개일 때는 초기값을 바꿔도 accuracy가 쉽게 높아지지 않았다. 따라서, 레이어가 3개일 때 더 sensitive하게 느껴졌다.

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.

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 at http://cs231n.github.io/neural-networks-3/#sgd for more information.

Open the file `cs231n/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.

In [8]:
```
from cs231n.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
```

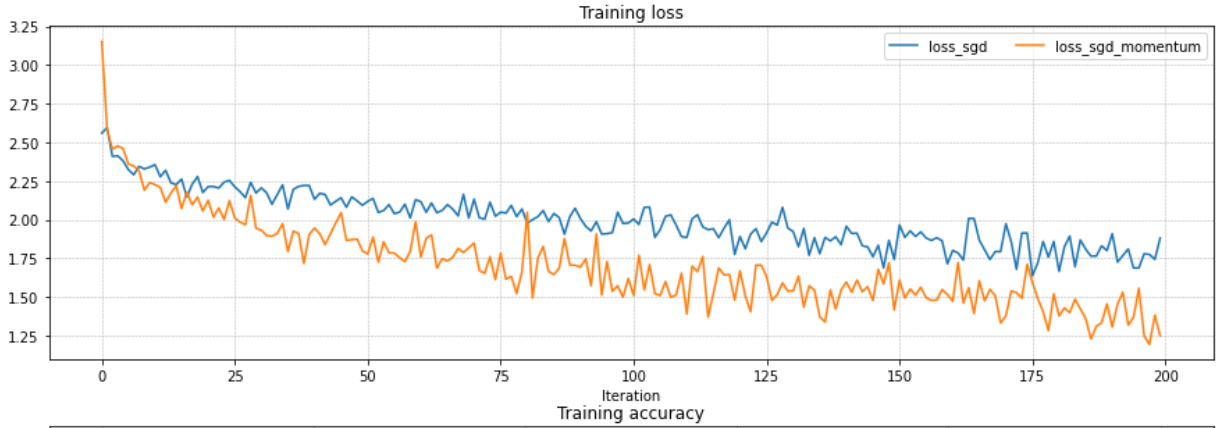
Once you have done so, run the following to train a six-layer network with both SGD and SGD+momentum. You should see the SGD+momentum update rule converge faster.

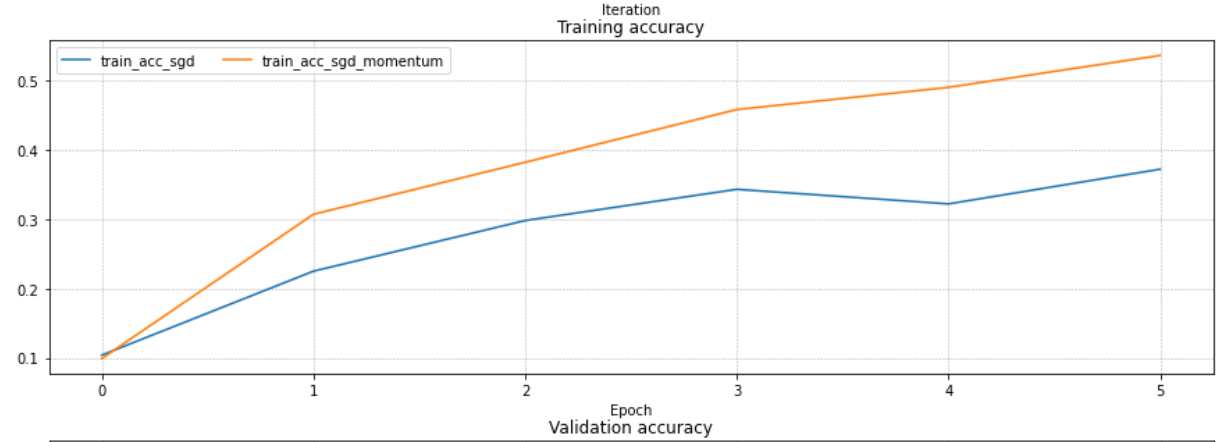
In [9]:
```
num_train = 4000
small_data = {
  'X_train': data['X_train'][:num_train],
  'y_train': data['y_train'][:num_train],
  'X_val': data['X_val'],
  'y_val': data['y_val'],
}

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,
    )
```

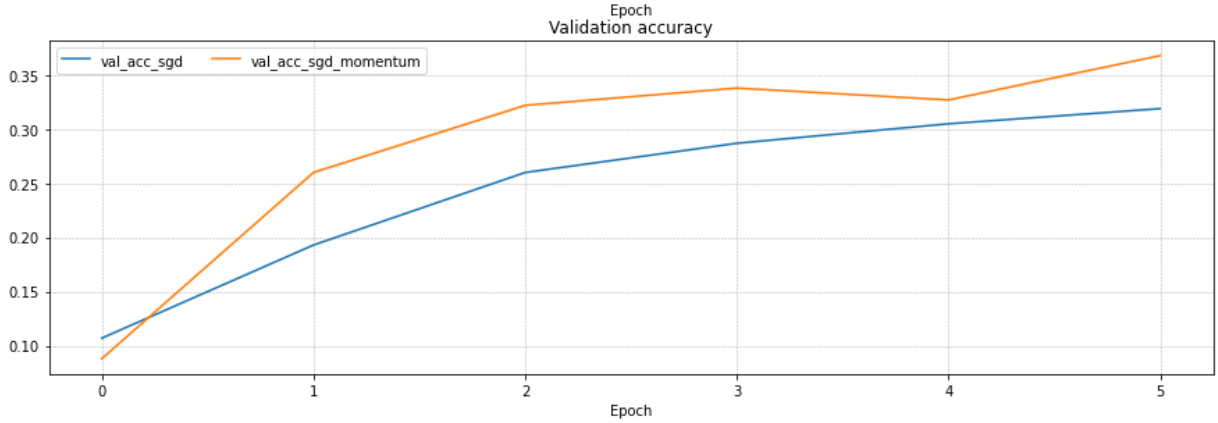
```
    solvers[update_rule] = solver
    solver.train()

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: 2.559978
(Epoch 0 / 5) train acc: 0.104000; val_acc: 0.107000
(Iteration 11 / 200) loss: 2.356069
(Iteration 21 / 200) loss: 2.214091
(Iteration 31 / 200) loss: 2.205928
(Epoch 1 / 5) train acc: 0.225000; val_acc: 0.193000
(Iteration 41 / 200) loss: 2.132095
(Iteration 51 / 200) loss: 2.118950
(Iteration 61 / 200) loss: 2.116443
(Iteration 71 / 200) loss: 2.132549
(Epoch 2 / 5) train acc: 0.298000; val_acc: 0.260000
(Iteration 81 / 200) loss: 1.977227
(Iteration 91 / 200) loss: 2.007528
(Iteration 101 / 200) loss: 2.004762
(Iteration 111 / 200) loss: 1.885342
(Epoch 3 / 5) train acc: 0.343000; val_acc: 0.287000
(Iteration 121 / 200) loss: 1.891516
(Iteration 131 / 200) loss: 1.923677
(Iteration 141 / 200) loss: 1.957743
(Iteration 151 / 200) loss: 1.966736
(Epoch 4 / 5) train acc: 0.322000; val_acc: 0.305000
(Iteration 161 / 200) loss: 1.801483
(Iteration 171 / 200) loss: 1.973780
(Iteration 181 / 200) loss: 1.666573
(Iteration 191 / 200) loss: 1.909494
(Epoch 5 / 5) train acc: 0.372000; val_acc: 0.319000
Running with  sgd_momentum
(Iteration 1 / 200) loss: 3.153778
(Epoch 0 / 5) train acc: 0.099000; val_acc: 0.088000
(Iteration 11 / 200) loss: 2.227203
(Iteration 21 / 200) loss: 2.125706
(Iteration 31 / 200) loss: 1.932695
(Epoch 1 / 5) train acc: 0.307000; val_acc: 0.260000
(Iteration 41 / 200) loss: 1.946488
(Iteration 51 / 200) loss: 1.778583
(Iteration 61 / 200) loss: 1.758119
(Iteration 71 / 200) loss: 1.849137
(Epoch 2 / 5) train acc: 0.382000; val_acc: 0.322000
(Iteration 81 / 200) loss: 2.048671
(Iteration 91 / 200) loss: 1.693223
(Iteration 101 / 200) loss: 1.511693
(Iteration 111 / 200) loss: 1.390754
(Epoch 3 / 5) train acc: 0.458000; val_acc: 0.338000
(Iteration 121 / 200) loss: 1.670614
```

```
(Iteration 131 / 200) loss: 1.540271
(Iteration 141 / 200) loss: 1.597365
(Iteration 151 / 200) loss: 1.609851
(Epoch 4 / 5) train acc: 0.490000; val_acc: 0.327000
(Iteration 161 / 200) loss: 1.472687
(Iteration 171 / 200) loss: 1.378620
(Iteration 181 / 200) loss: 1.378175
(Iteration 191 / 200) loss: 1.305934
(Epoch 5 / 5) train acc: 0.536000; val_acc: 0.368000
```







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 `cs231n/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.

In [10]:
```
# Test RMSProp implementation
from cs231n.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
```

In [11]:
```
# Test Adam implementation
from cs231n.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:  1.139887467333134e-07
v error:  4.208314038113071e-09
m error:  4.214963193114416e-09
```

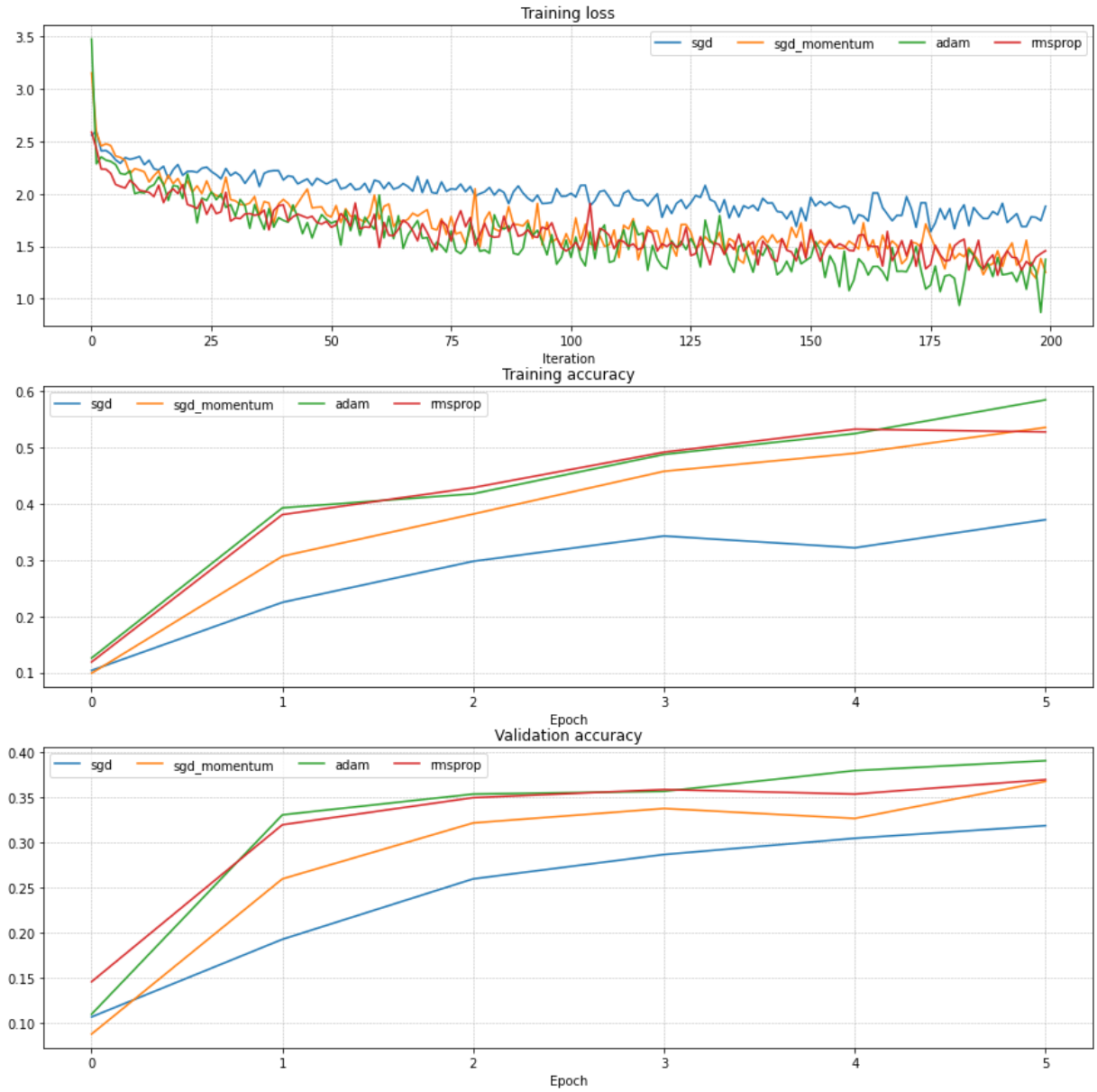
Once you have debugged your RMSProp and Adam implementations, run the following to train a pair of deep networks using these new update rules:

In [12]:

```
learning_rates = {'rmsprop': 1e-4, 'adam': 1e-3}
for update_rule in ['adam', 'rmsprop']:
    print('Running with ', update_rule)
    model = FullyConnectedNet(
        [100, 100, 100, 100, 100],
        weight_scale=5e-2
    )
    solver = Solver(
        model,
        small_data,
        num_epochs=5,
        batch_size=100,
        update_rule=update_rule,
        optim_config={'learning_rate': learning_rates[update_rule]},
        verbose=True
    )
    solvers[update_rule] = solver
    solver.train()
    print()

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

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

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

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

plt.show()
```

```
Running with  adam
(Iteration 1 / 200) loss: 3.476928
(Epoch 0 / 5) train acc: 0.126000; val_acc: 0.110000
(Iteration 11 / 200) loss: 2.015214
(Iteration 21 / 200) loss: 2.190157
(Iteration 31 / 200) loss: 1.785010
(Epoch 1 / 5) train acc: 0.393000; val_acc: 0.331000
(Iteration 41 / 200) loss: 1.746502
(Iteration 51 / 200) loss: 1.719389
(Iteration 61 / 200) loss: 1.988491
(Iteration 71 / 200) loss: 1.582892
(Epoch 2 / 5) train acc: 0.418000; val_acc: 0.354000
(Iteration 81 / 200) loss: 1.593839
(Iteration 91 / 200) loss: 1.479027
```

```
(Iteration 101 / 200) loss: 1.393083
(Iteration 111 / 200) loss: 1.527003
(Epoch 3 / 5) train acc: 0.488000; val_acc: 0.357000
(Iteration 121 / 200) loss: 1.282885
(Iteration 131 / 200) loss: 1.505785
(Iteration 141 / 200) loss: 1.411430
(Iteration 151 / 200) loss: 1.395196
(Epoch 4 / 5) train acc: 0.525000; val_acc: 0.380000
(Iteration 161 / 200) loss: 1.380678
(Iteration 171 / 200) loss: 1.257138
(Iteration 181 / 200) loss: 1.192556
(Iteration 191 / 200) loss: 1.226958
(Epoch 5 / 5) train acc: 0.585000; val_acc: 0.391000

Running with  rmsprop
(Iteration 1 / 200) loss: 2.589166
(Epoch 0 / 5) train acc: 0.119000; val_acc: 0.146000
(Iteration 11 / 200) loss: 2.032921
(Iteration 21 / 200) loss: 1.897278
(Iteration 31 / 200) loss: 1.770793
(Epoch 1 / 5) train acc: 0.381000; val_acc: 0.320000
(Iteration 41 / 200) loss: 1.895731
(Iteration 51 / 200) loss: 1.681091
(Iteration 61 / 200) loss: 1.487204
(Iteration 71 / 200) loss: 1.629973
(Epoch 2 / 5) train acc: 0.429000; val_acc: 0.350000
(Iteration 81 / 200) loss: 1.506686
(Iteration 91 / 200) loss: 1.610742
(Iteration 101 / 200) loss: 1.486124
(Iteration 111 / 200) loss: 1.559454
(Epoch 3 / 5) train acc: 0.492000; val_acc: 0.359000
(Iteration 121 / 200) loss: 1.496860
(Iteration 131 / 200) loss: 1.531552
(Iteration 141 / 200) loss: 1.550195
(Iteration 151 / 200) loss: 1.657838
(Epoch 4 / 5) train acc: 0.533000; val_acc: 0.354000
(Iteration 161 / 200) loss: 1.603105
(Iteration 171 / 200) loss: 1.408064
(Iteration 181 / 200) loss: 1.504707
(Iteration 191 / 200) loss: 1.385212
(Epoch 5 / 5) train acc: 0.528000; val_acc: 0.370000
```



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?

Answer:

수식을 주목하자. 수식을 보면 cache의 제곱근이 분모로 계속 들어가는 것을 볼 수 있다. iteration이 진행되다 보면 이는 누적적으로 계속 cache의 값이 늘어갈 것이고, 그러면 분모가 매우 커지게 되어 update가 매우 작아지게 될 것이다. Adam의 경우에는 exponential weighted average를 사용하기 때문에 이러한 문제가 발생하지 않는다.

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.

If you are careful it should be possible to get accuracies above 55%, but we don't require it for this part and won't assign extra credit for doing so. Later in the assignment we will ask you to train the best convolutional network that you can on CIFAR-10, and we would prefer that you spend your effort working on convolutional networks rather than fully connected networks.

Note: You might find it useful to complete the `BatchNormalization.ipynb` and `Dropout.ipynb` notebooks before completing this part, since those techniques can help you train powerful models.

In [47]:
```
best_model = None

################################################################################
# TODO: Train the best FullyConnectedNet that you can on CIFAR-10. You might   #
# find batch/layer normalization and dropout useful. Store your best model in  #
# the best_model variable.                                                     #
################################################################################
# *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****

from itertools import product

best_val_acc = 0


learning_rates = [1e-3,1e-4]
weight_scales = [1e-2,5e-2,1e-3]
grid_search = list(product(*[learning_rates, weight_scales]))

for lr, ws in grid_search :
  model = FullyConnectedNet([100,100,100,100,100], weight_scale = ws)
  solver = Solver(model, data, num_epochs = 10, batch_size = 200,
                  update_rule = 'adam',
                  optim_config = {'learning_rate' : lr})
  solver.train()

  if solver.best_val_acc > best_val_acc :
    best_model = model
    best_val_acc = solver.best_val_acc
    best_lr = lr
    best_ws = ws

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

```
(Iteration 1 / 2450) loss: 2.302585
(Epoch 0 / 10) train acc: 0.115000; val_acc: 0.099000
(Iteration 11 / 2450) loss: 2.302620
(Iteration 21 / 2450) loss: 2.287324
(Iteration 31 / 2450) loss: 2.219808
(Iteration 41 / 2450) loss: 2.051530
(Iteration 51 / 2450) loss: 2.087850
```

```
(Iteration 61 / 2450) loss: 2.065916
(Iteration 71 / 2450) loss: 2.078800
(Iteration 81 / 2450) loss: 2.039025
(Iteration 91 / 2450) loss: 1.999872
(Iteration 101 / 2450) loss: 2.010886
(Iteration 111 / 2450) loss: 2.020884
(Iteration 121 / 2450) loss: 1.981685
(Iteration 131 / 2450) loss: 2.031360
(Iteration 141 / 2450) loss: 2.035490
(Iteration 151 / 2450) loss: 1.960708
(Iteration 161 / 2450) loss: 1.921058
(Iteration 171 / 2450) loss: 2.023753
(Iteration 181 / 2450) loss: 2.069451
(Iteration 191 / 2450) loss: 1.898963
(Iteration 201 / 2450) loss: 1.895850
(Iteration 211 / 2450) loss: 1.931971
(Iteration 221 / 2450) loss: 1.866638
(Iteration 231 / 2450) loss: 1.866226
(Iteration 241 / 2450) loss: 1.811044
(Epoch 1 / 10) train acc: 0.280000; val_acc: 0.290000
(Iteration 251 / 2450) loss: 1.808385
(Iteration 261 / 2450) loss: 1.923047
(Iteration 271 / 2450) loss: 1.798686
(Iteration 281 / 2450) loss: 1.838230
(Iteration 291 / 2450) loss: 1.810238
(Iteration 301 / 2450) loss: 1.969841
(Iteration 311 / 2450) loss: 1.781761
(Iteration 321 / 2450) loss: 1.803562
(Iteration 331 / 2450) loss: 1.766733
(Iteration 341 / 2450) loss: 1.868737
(Iteration 351 / 2450) loss: 1.808128
(Iteration 361 / 2450) loss: 1.771489
(Iteration 371 / 2450) loss: 1.648002
(Iteration 381 / 2450) loss: 1.758546
(Iteration 391 / 2450) loss: 1.640638
(Iteration 401 / 2450) loss: 1.636884
(Iteration 411 / 2450) loss: 1.698905
(Iteration 421 / 2450) loss: 1.789568
(Iteration 431 / 2450) loss: 1.759323
(Iteration 441 / 2450) loss: 1.789457
(Iteration 451 / 2450) loss: 1.749889
(Iteration 461 / 2450) loss: 1.625453
(Iteration 471 / 2450) loss: 1.495929
(Iteration 481 / 2450) loss: 1.579466
(Epoch 2 / 10) train acc: 0.386000; val_acc: 0.407000
(Iteration 491 / 2450) loss: 1.538419
(Iteration 501 / 2450) loss: 1.633497
(Iteration 511 / 2450) loss: 1.604955
(Iteration 521 / 2450) loss: 1.582791
(Iteration 531 / 2450) loss: 1.588058
(Iteration 541 / 2450) loss: 1.669825
(Iteration 551 / 2450) loss: 1.551945
(Iteration 561 / 2450) loss: 1.714737
(Iteration 571 / 2450) loss: 1.604463
(Iteration 581 / 2450) loss: 1.490939
(Iteration 591 / 2450) loss: 1.499496
(Iteration 601 / 2450) loss: 1.583745
(Iteration 611 / 2450) loss: 1.567865
(Iteration 621 / 2450) loss: 1.534580
(Iteration 631 / 2450) loss: 1.552479
(Iteration 641 / 2450) loss: 1.678869
(Iteration 651 / 2450) loss: 1.593236
(Iteration 661 / 2450) loss: 1.384041
(Iteration 671 / 2450) loss: 1.525958
(Iteration 681 / 2450) loss: 1.532739
(Iteration 691 / 2450) loss: 1.561754
(Iteration 701 / 2450) loss: 1.368937
(Iteration 711 / 2450) loss: 1.530145
(Iteration 721 / 2450) loss: 1.611456
```

```
(Iteration 731 / 2450) loss: 1.522010
(Epoch 3 / 10) train acc: 0.467000; val_acc: 0.461000
(Iteration 741 / 2450) loss: 1.463302
(Iteration 751 / 2450) loss: 1.589906
(Iteration 761 / 2450) loss: 1.535857
(Iteration 771 / 2450) loss: 1.386532
(Iteration 781 / 2450) loss: 1.487269
(Iteration 791 / 2450) loss: 1.416020
(Iteration 801 / 2450) loss: 1.443347
(Iteration 811 / 2450) loss: 1.488077
(Iteration 821 / 2450) loss: 1.416701
(Iteration 831 / 2450) loss: 1.355539
(Iteration 841 / 2450) loss: 1.478343
(Iteration 851 / 2450) loss: 1.471000
(Iteration 861 / 2450) loss: 1.469756
(Iteration 871 / 2450) loss: 1.411971
(Iteration 881 / 2450) loss: 1.558202
(Iteration 891 / 2450) loss: 1.497883
(Iteration 901 / 2450) loss: 1.417722
(Iteration 911 / 2450) loss: 1.512253
(Iteration 921 / 2450) loss: 1.496712
(Iteration 931 / 2450) loss: 1.437772
(Iteration 941 / 2450) loss: 1.399766
(Iteration 951 / 2450) loss: 1.466614
(Iteration 961 / 2450) loss: 1.399659
(Iteration 971 / 2450) loss: 1.415035
(Epoch 4 / 10) train acc: 0.522000; val_acc: 0.485000
(Iteration 981 / 2450) loss: 1.455471
(Iteration 991 / 2450) loss: 1.514571
(Iteration 1001 / 2450) loss: 1.386758
(Iteration 1011 / 2450) loss: 1.308116
(Iteration 1021 / 2450) loss: 1.363413
(Iteration 1031 / 2450) loss: 1.449274
(Iteration 1041 / 2450) loss: 1.289611
(Iteration 1051 / 2450) loss: 1.252978
(Iteration 1061 / 2450) loss: 1.259132
(Iteration 1071 / 2450) loss: 1.428065
(Iteration 1081 / 2450) loss: 1.275640
(Iteration 1091 / 2450) loss: 1.453197
(Iteration 1101 / 2450) loss: 1.275018
(Iteration 1111 / 2450) loss: 1.308759
(Iteration 1121 / 2450) loss: 1.206442
(Iteration 1131 / 2450) loss: 1.259742
(Iteration 1141 / 2450) loss: 1.355821
(Iteration 1151 / 2450) loss: 1.337534
(Iteration 1161 / 2450) loss: 1.239572
(Iteration 1171 / 2450) loss: 1.293608
(Iteration 1181 / 2450) loss: 1.319696
(Iteration 1191 / 2450) loss: 1.342169
(Iteration 1201 / 2450) loss: 1.462655
(Iteration 1211 / 2450) loss: 1.248791
(Iteration 1221 / 2450) loss: 1.334770
(Epoch 5 / 10) train acc: 0.506000; val_acc: 0.460000
(Iteration 1231 / 2450) loss: 1.345129
(Iteration 1241 / 2450) loss: 1.286996
(Iteration 1251 / 2450) loss: 1.315349
(Iteration 1261 / 2450) loss: 1.308363
(Iteration 1271 / 2450) loss: 1.244395
(Iteration 1281 / 2450) loss: 1.252506
(Iteration 1291 / 2450) loss: 1.399638
(Iteration 1301 / 2450) loss: 1.356939
(Iteration 1311 / 2450) loss: 1.442439
(Iteration 1321 / 2450) loss: 1.368392
(Iteration 1331 / 2450) loss: 1.399702
(Iteration 1341 / 2450) loss: 1.314315
(Iteration 1351 / 2450) loss: 1.151054
(Iteration 1361 / 2450) loss: 1.265466
(Iteration 1371 / 2450) loss: 1.392918
(Iteration 1381 / 2450) loss: 1.276088
```

```
(Iteration 1391 / 2450) loss: 1.316322
(Iteration 1401 / 2450) loss: 1.263900
(Iteration 1411 / 2450) loss: 1.300898
(Iteration 1421 / 2450) loss: 1.250033
(Iteration 1431 / 2450) loss: 1.266996
(Iteration 1441 / 2450) loss: 1.312010
(Iteration 1451 / 2450) loss: 1.286484
(Iteration 1461 / 2450) loss: 1.382204
(Epoch 6 / 10) train acc: 0.542000; val_acc: 0.489000
(Iteration 1471 / 2450) loss: 1.325096
(Iteration 1481 / 2450) loss: 1.199817
(Iteration 1491 / 2450) loss: 1.207212
(Iteration 1501 / 2450) loss: 1.246337
(Iteration 1511 / 2450) loss: 1.264467
(Iteration 1521 / 2450) loss: 1.243342
(Iteration 1531 / 2450) loss: 1.223897
(Iteration 1541 / 2450) loss: 1.199580
(Iteration 1551 / 2450) loss: 1.211664
(Iteration 1561 / 2450) loss: 1.234069
(Iteration 1571 / 2450) loss: 1.307475
(Iteration 1581 / 2450) loss: 1.180783
(Iteration 1591 / 2450) loss: 1.215374
(Iteration 1601 / 2450) loss: 1.349473
(Iteration 1611 / 2450) loss: 1.164671
(Iteration 1621 / 2450) loss: 1.198818
(Iteration 1631 / 2450) loss: 1.190600
(Iteration 1641 / 2450) loss: 1.210217
(Iteration 1651 / 2450) loss: 1.218504
(Iteration 1661 / 2450) loss: 1.188403
(Iteration 1671 / 2450) loss: 1.234212
(Iteration 1681 / 2450) loss: 1.207505
(Iteration 1691 / 2450) loss: 1.300782
(Iteration 1701 / 2450) loss: 1.151364
(Iteration 1711 / 2450) loss: 1.277723
(Epoch 7 / 10) train acc: 0.576000; val_acc: 0.510000
(Iteration 1721 / 2450) loss: 1.208320
(Iteration 1731 / 2450) loss: 1.367549
(Iteration 1741 / 2450) loss: 1.323054
(Iteration 1751 / 2450) loss: 1.124364
(Iteration 1761 / 2450) loss: 1.205913
(Iteration 1771 / 2450) loss: 1.124188
(Iteration 1781 / 2450) loss: 1.259404
(Iteration 1791 / 2450) loss: 1.234593
(Iteration 1801 / 2450) loss: 1.276401
(Iteration 1811 / 2450) loss: 1.132993
(Iteration 1821 / 2450) loss: 1.229565
(Iteration 1831 / 2450) loss: 1.333016
(Iteration 1841 / 2450) loss: 1.250982
(Iteration 1851 / 2450) loss: 1.173201
(Iteration 1861 / 2450) loss: 1.145381
(Iteration 1871 / 2450) loss: 1.285885
(Iteration 1881 / 2450) loss: 1.125260
(Iteration 1891 / 2450) loss: 1.223742
(Iteration 1901 / 2450) loss: 1.378496
(Iteration 1911 / 2450) loss: 1.200304
(Iteration 1921 / 2450) loss: 1.213217
(Iteration 1931 / 2450) loss: 1.148116
(Iteration 1941 / 2450) loss: 1.160461
(Iteration 1951 / 2450) loss: 1.324437
(Epoch 8 / 10) train acc: 0.571000; val_acc: 0.509000
(Iteration 1961 / 2450) loss: 1.100900
(Iteration 1971 / 2450) loss: 1.209427
(Iteration 1981 / 2450) loss: 1.142645
(Iteration 1991 / 2450) loss: 1.221889
(Iteration 2001 / 2450) loss: 1.075270
(Iteration 2011 / 2450) loss: 1.091918
(Iteration 2021 / 2450) loss: 1.214684
(Iteration 2031 / 2450) loss: 1.189536
(Iteration 2041 / 2450) loss: 1.157045
```

```
(Iteration 2051 / 2450) loss: 1.123956
(Iteration 2061 / 2450) loss: 1.105427
(Iteration 2071 / 2450) loss: 1.243732
(Iteration 2081 / 2450) loss: 1.212954
(Iteration 2091 / 2450) loss: 1.211802
(Iteration 2101 / 2450) loss: 1.147832
(Iteration 2111 / 2450) loss: 1.068337
(Iteration 2121 / 2450) loss: 1.108692
(Iteration 2131 / 2450) loss: 1.269148
(Iteration 2141 / 2450) loss: 1.213601
(Iteration 2151 / 2450) loss: 1.089706
(Iteration 2161 / 2450) loss: 1.099062
(Iteration 2171 / 2450) loss: 1.030133
(Iteration 2181 / 2450) loss: 1.211046
(Iteration 2191 / 2450) loss: 1.338480
(Iteration 2201 / 2450) loss: 1.230711
(Epoch 9 / 10) train acc: 0.596000; val_acc: 0.532000
(Iteration 2211 / 2450) loss: 1.174168
(Iteration 2221 / 2450) loss: 1.181849
(Iteration 2231 / 2450) loss: 1.209290
(Iteration 2241 / 2450) loss: 1.139722
(Iteration 2251 / 2450) loss: 1.055957
(Iteration 2261 / 2450) loss: 1.173584
(Iteration 2271 / 2450) loss: 1.165867
(Iteration 2281 / 2450) loss: 1.142762
(Iteration 2291 / 2450) loss: 1.151893
(Iteration 2301 / 2450) loss: 1.190000
(Iteration 2311 / 2450) loss: 1.082070
(Iteration 2321 / 2450) loss: 1.233615
(Iteration 2331 / 2450) loss: 1.208518
(Iteration 2341 / 2450) loss: 1.084146
(Iteration 2351 / 2450) loss: 1.222026
(Iteration 2361 / 2450) loss: 1.202698
(Iteration 2371 / 2450) loss: 0.995536
(Iteration 2381 / 2450) loss: 1.141984
(Iteration 2391 / 2450) loss: 1.290401
(Iteration 2401 / 2450) loss: 1.024354
(Iteration 2411 / 2450) loss: 1.204694
(Iteration 2421 / 2450) loss: 1.073487
(Iteration 2431 / 2450) loss: 0.980983
(Iteration 2441 / 2450) loss: 1.250445
(Epoch 10 / 10) train acc: 0.599000; val_acc: 0.510000
(Iteration 1 / 2450) loss: 2.751240
(Epoch 0 / 10) train acc: 0.134000; val_acc: 0.134000
(Iteration 11 / 2450) loss: 2.070501
(Iteration 21 / 2450) loss: 2.036511
(Iteration 31 / 2450) loss: 2.042558
(Iteration 41 / 2450) loss: 1.803510
(Iteration 51 / 2450) loss: 1.775050
(Iteration 61 / 2450) loss: 1.864572
(Iteration 71 / 2450) loss: 1.836567
(Iteration 81 / 2450) loss: 1.657693
(Iteration 91 / 2450) loss: 1.704371
(Iteration 101 / 2450) loss: 1.683435
(Iteration 111 / 2450) loss: 1.701612
(Iteration 121 / 2450) loss: 1.698064
(Iteration 131 / 2450) loss: 1.586945
(Iteration 141 / 2450) loss: 1.617715
(Iteration 151 / 2450) loss: 1.739003
(Iteration 161 / 2450) loss: 1.683557
(Iteration 171 / 2450) loss: 1.561320
(Iteration 181 / 2450) loss: 1.591565
(Iteration 191 / 2450) loss: 1.525484
(Iteration 201 / 2450) loss: 1.452969
(Iteration 211 / 2450) loss: 1.826542
(Iteration 221 / 2450) loss: 1.507271
(Iteration 231 / 2450) loss: 1.658104
(Iteration 241 / 2450) loss: 1.526162
(Epoch 1 / 10) train acc: 0.426000; val_acc: 0.425000
```

```
(Iteration 251 / 2450) loss: 1.623143
(Iteration 261 / 2450) loss: 1.571702
(Iteration 271 / 2450) loss: 1.550407
(Iteration 281 / 2450) loss: 1.519835
(Iteration 291 / 2450) loss: 1.620129
(Iteration 301 / 2450) loss: 1.523275
(Iteration 311 / 2450) loss: 1.515101
(Iteration 321 / 2450) loss: 1.485442
(Iteration 331 / 2450) loss: 1.519223
(Iteration 341 / 2450) loss: 1.414627
(Iteration 351 / 2450) loss: 1.517677
(Iteration 361 / 2450) loss: 1.600407
(Iteration 371 / 2450) loss: 1.564065
(Iteration 381 / 2450) loss: 1.491324
(Iteration 391 / 2450) loss: 1.605604
(Iteration 401 / 2450) loss: 1.581190
(Iteration 411 / 2450) loss: 1.478043
(Iteration 421 / 2450) loss: 1.415786
(Iteration 431 / 2450) loss: 1.388014
(Iteration 441 / 2450) loss: 1.509650
(Iteration 451 / 2450) loss: 1.422245
(Iteration 461 / 2450) loss: 1.393798
(Iteration 471 / 2450) loss: 1.489749
(Iteration 481 / 2450) loss: 1.546096
(Epoch 2 / 10) train acc: 0.462000; val_acc: 0.468000
(Iteration 491 / 2450) loss: 1.461981
(Iteration 501 / 2450) loss: 1.618322
(Iteration 511 / 2450) loss: 1.493725
(Iteration 521 / 2450) loss: 1.446319
(Iteration 531 / 2450) loss: 1.510450
(Iteration 541 / 2450) loss: 1.370656
(Iteration 551 / 2450) loss: 1.429112
(Iteration 561 / 2450) loss: 1.420148
(Iteration 571 / 2450) loss: 1.537726
(Iteration 581 / 2450) loss: 1.387966
(Iteration 591 / 2450) loss: 1.318160
(Iteration 601 / 2450) loss: 1.491891
(Iteration 611 / 2450) loss: 1.304308
(Iteration 621 / 2450) loss: 1.485269
(Iteration 631 / 2450) loss: 1.379513
(Iteration 641 / 2450) loss: 1.345627
(Iteration 651 / 2450) loss: 1.480225
(Iteration 661 / 2450) loss: 1.489315
(Iteration 671 / 2450) loss: 1.322890
(Iteration 681 / 2450) loss: 1.354877
(Iteration 691 / 2450) loss: 1.346313
(Iteration 701 / 2450) loss: 1.392564
(Iteration 711 / 2450) loss: 1.390622
(Iteration 721 / 2450) loss: 1.415497
(Iteration 731 / 2450) loss: 1.390479
(Epoch 3 / 10) train acc: 0.499000; val_acc: 0.472000
(Iteration 741 / 2450) loss: 1.370629
(Iteration 751 / 2450) loss: 1.493760
(Iteration 761 / 2450) loss: 1.301374
(Iteration 771 / 2450) loss: 1.451522
(Iteration 781 / 2450) loss: 1.470959
(Iteration 791 / 2450) loss: 1.475197
(Iteration 801 / 2450) loss: 1.529353
(Iteration 811 / 2450) loss: 1.327499
(Iteration 821 / 2450) loss: 1.367698
(Iteration 831 / 2450) loss: 1.428444
(Iteration 841 / 2450) loss: 1.261870
(Iteration 851 / 2450) loss: 1.287452
(Iteration 861 / 2450) loss: 1.386638
(Iteration 871 / 2450) loss: 1.332971
(Iteration 881 / 2450) loss: 1.302521
(Iteration 891 / 2450) loss: 1.290978
(Iteration 901 / 2450) loss: 1.295396
(Iteration 911 / 2450) loss: 1.242574
```

```
(Iteration 921 / 2450) loss: 1.141145
(Iteration 931 / 2450) loss: 1.335693
(Iteration 941 / 2450) loss: 1.351004
(Iteration 951 / 2450) loss: 1.247007
(Iteration 961 / 2450) loss: 1.276175
(Iteration 971 / 2450) loss: 1.299852
(Epoch 4 / 10) train acc: 0.531000; val_acc: 0.475000
(Iteration 981 / 2450) loss: 1.425078
(Iteration 991 / 2450) loss: 1.246413
(Iteration 1001 / 2450) loss: 1.314146
(Iteration 1011 / 2450) loss: 1.332281
(Iteration 1021 / 2450) loss: 1.268748
(Iteration 1031 / 2450) loss: 1.149184
(Iteration 1041 / 2450) loss: 1.335075
(Iteration 1051 / 2450) loss: 1.307773
(Iteration 1061 / 2450) loss: 1.280112
(Iteration 1071 / 2450) loss: 1.156845
(Iteration 1081 / 2450) loss: 1.215046
(Iteration 1091 / 2450) loss: 1.134798
(Iteration 1101 / 2450) loss: 1.286191
(Iteration 1111 / 2450) loss: 1.220781
(Iteration 1121 / 2450) loss: 1.287689
(Iteration 1131 / 2450) loss: 1.384487
(Iteration 1141 / 2450) loss: 1.277492
(Iteration 1151 / 2450) loss: 1.192436
(Iteration 1161 / 2450) loss: 1.247837
(Iteration 1171 / 2450) loss: 1.192000
(Iteration 1181 / 2450) loss: 1.317150
(Iteration 1191 / 2450) loss: 1.334860
(Iteration 1201 / 2450) loss: 1.304023
(Iteration 1211 / 2450) loss: 1.320823
(Iteration 1221 / 2450) loss: 1.248810
(Epoch 5 / 10) train acc: 0.565000; val_acc: 0.500000
(Iteration 1231 / 2450) loss: 1.184423
(Iteration 1241 / 2450) loss: 1.148900
(Iteration 1251 / 2450) loss: 1.182513
(Iteration 1261 / 2450) loss: 1.297840
(Iteration 1271 / 2450) loss: 1.054710
(Iteration 1281 / 2450) loss: 1.190441
(Iteration 1291 / 2450) loss: 1.206431
(Iteration 1301 / 2450) loss: 1.320105
(Iteration 1311 / 2450) loss: 1.182751
(Iteration 1321 / 2450) loss: 1.199502
(Iteration 1331 / 2450) loss: 1.193467
(Iteration 1341 / 2450) loss: 1.287509
(Iteration 1351 / 2450) loss: 1.193040
(Iteration 1361 / 2450) loss: 1.217269
(Iteration 1371 / 2450) loss: 1.134198
(Iteration 1381 / 2450) loss: 1.241719
(Iteration 1391 / 2450) loss: 1.244760
(Iteration 1401 / 2450) loss: 1.324115
(Iteration 1411 / 2450) loss: 1.325721
(Iteration 1421 / 2450) loss: 1.270231
(Iteration 1431 / 2450) loss: 1.223808
(Iteration 1441 / 2450) loss: 1.283505
(Iteration 1451 / 2450) loss: 1.381239
(Iteration 1461 / 2450) loss: 1.147316
(Epoch 6 / 10) train acc: 0.598000; val_acc: 0.508000
(Iteration 1471 / 2450) loss: 1.240146
(Iteration 1481 / 2450) loss: 1.201414
(Iteration 1491 / 2450) loss: 1.250626
(Iteration 1501 / 2450) loss: 1.084272
(Iteration 1511 / 2450) loss: 1.237742
(Iteration 1521 / 2450) loss: 1.235046
(Iteration 1531 / 2450) loss: 1.252793
(Iteration 1541 / 2450) loss: 1.317638
(Iteration 1551 / 2450) loss: 1.184035
(Iteration 1561 / 2450) loss: 1.108049
(Iteration 1571 / 2450) loss: 1.291723
```

```
(Iteration 1581 / 2450) loss: 1.148888
(Iteration 1591 / 2450) loss: 1.054504
(Iteration 1601 / 2450) loss: 1.208677
(Iteration 1611 / 2450) loss: 1.256847
(Iteration 1621 / 2450) loss: 1.133033
(Iteration 1631 / 2450) loss: 1.286153
(Iteration 1641 / 2450) loss: 1.154937
(Iteration 1651 / 2450) loss: 1.238932
(Iteration 1661 / 2450) loss: 1.037288
(Iteration 1671 / 2450) loss: 1.118699
(Iteration 1681 / 2450) loss: 1.195628
(Iteration 1691 / 2450) loss: 1.104774
(Iteration 1701 / 2450) loss: 1.092302
(Iteration 1711 / 2450) loss: 1.190425
(Epoch 7 / 10) train acc: 0.578000; val_acc: 0.510000
(Iteration 1721 / 2450) loss: 1.173357
(Iteration 1731 / 2450) loss: 1.244647
(Iteration 1741 / 2450) loss: 1.261131
(Iteration 1751 / 2450) loss: 1.083790
(Iteration 1761 / 2450) loss: 1.112181
(Iteration 1771 / 2450) loss: 1.122199
(Iteration 1781 / 2450) loss: 1.106056
(Iteration 1791 / 2450) loss: 1.057796
(Iteration 1801 / 2450) loss: 1.028983
(Iteration 1811 / 2450) loss: 1.131944
(Iteration 1821 / 2450) loss: 1.308831
(Iteration 1831 / 2450) loss: 1.071855
(Iteration 1841 / 2450) loss: 0.943782
(Iteration 1851 / 2450) loss: 1.267668
(Iteration 1861 / 2450) loss: 1.187620
(Iteration 1871 / 2450) loss: 1.262734
(Iteration 1881 / 2450) loss: 1.177885
(Iteration 1891 / 2450) loss: 1.177654
(Iteration 1901 / 2450) loss: 1.013667
(Iteration 1911 / 2450) loss: 1.175474
(Iteration 1921 / 2450) loss: 1.112511
(Iteration 1931 / 2450) loss: 1.243013
(Iteration 1941 / 2450) loss: 1.085442
(Iteration 1951 / 2450) loss: 1.120114
(Epoch 8 / 10) train acc: 0.579000; val_acc: 0.516000
(Iteration 1961 / 2450) loss: 1.106055
(Iteration 1971 / 2450) loss: 1.093205
(Iteration 1981 / 2450) loss: 1.024166
(Iteration 1991 / 2450) loss: 1.281745
(Iteration 2001 / 2450) loss: 1.245367
(Iteration 2011 / 2450) loss: 1.099036
(Iteration 2021 / 2450) loss: 1.168409
(Iteration 2031 / 2450) loss: 1.128444
(Iteration 2041 / 2450) loss: 1.090186
(Iteration 2051 / 2450) loss: 1.233793
(Iteration 2061 / 2450) loss: 1.174334
(Iteration 2071 / 2450) loss: 1.001767
(Iteration 2081 / 2450) loss: 1.183767
(Iteration 2091 / 2450) loss: 1.217057
(Iteration 2101 / 2450) loss: 1.148039
(Iteration 2111 / 2450) loss: 1.108658
(Iteration 2121 / 2450) loss: 1.113836
(Iteration 2131 / 2450) loss: 1.213707
(Iteration 2141 / 2450) loss: 1.191151
(Iteration 2151 / 2450) loss: 1.175773
(Iteration 2161 / 2450) loss: 1.072329
(Iteration 2171 / 2450) loss: 1.094342
(Iteration 2181 / 2450) loss: 1.232995
(Iteration 2191 / 2450) loss: 1.049244
(Iteration 2201 / 2450) loss: 1.005511
(Epoch 9 / 10) train acc: 0.577000; val_acc: 0.530000
(Iteration 2211 / 2450) loss: 0.870410
(Iteration 2221 / 2450) loss: 1.093910
(Iteration 2231 / 2450) loss: 1.167180
```

```
(Iteration 2241 / 2450) loss: 1.104838
(Iteration 2251 / 2450) loss: 1.045142
(Iteration 2261 / 2450) loss: 1.141494
(Iteration 2271 / 2450) loss: 1.191022
(Iteration 2281 / 2450) loss: 1.199994
(Iteration 2291 / 2450) loss: 0.980059
(Iteration 2301 / 2450) loss: 1.007437
(Iteration 2311 / 2450) loss: 1.224169
(Iteration 2321 / 2450) loss: 1.070709
(Iteration 2331 / 2450) loss: 1.028882
(Iteration 2341 / 2450) loss: 0.959052
(Iteration 2351 / 2450) loss: 1.041466
(Iteration 2361 / 2450) loss: 1.181918
(Iteration 2371 / 2450) loss: 0.905507
(Iteration 2381 / 2450) loss: 1.084210
(Iteration 2391 / 2450) loss: 1.127327
(Iteration 2401 / 2450) loss: 1.111947
(Iteration 2411 / 2450) loss: 1.183992
(Iteration 2421 / 2450) loss: 1.069674
(Iteration 2431 / 2450) loss: 1.050056
(Iteration 2441 / 2450) loss: 1.200116
(Epoch 10 / 10) train acc: 0.620000; val_acc: 0.502000
(Iteration 1 / 2450) loss: 2.302585
(Epoch 0 / 10) train acc: 0.093000; val_acc: 0.119000
(Iteration 11 / 2450) loss: 2.303147
(Iteration 21 / 2450) loss: 2.303571
(Iteration 31 / 2450) loss: 2.302583
(Iteration 41 / 2450) loss: 2.303255
(Iteration 51 / 2450) loss: 2.303523
(Iteration 61 / 2450) loss: 2.301904
(Iteration 71 / 2450) loss: 2.302637
(Iteration 81 / 2450) loss: 2.302069
(Iteration 91 / 2450) loss: 2.302547
(Iteration 101 / 2450) loss: 2.302699
(Iteration 111 / 2450) loss: 2.302731
(Iteration 121 / 2450) loss: 2.302142
(Iteration 131 / 2450) loss: 2.303294
(Iteration 141 / 2450) loss: 2.303182
(Iteration 151 / 2450) loss: 2.303063
(Iteration 161 / 2450) loss: 2.302075
(Iteration 171 / 2450) loss: 2.302812
(Iteration 181 / 2450) loss: 2.302259
(Iteration 191 / 2450) loss: 2.301738
(Iteration 201 / 2450) loss: 2.302598
(Iteration 211 / 2450) loss: 2.302295
(Iteration 221 / 2450) loss: 2.303685
(Iteration 231 / 2450) loss: 2.302266
(Iteration 241 / 2450) loss: 2.303388
(Epoch 1 / 10) train acc: 0.094000; val_acc: 0.113000
(Iteration 251 / 2450) loss: 2.301953
(Iteration 261 / 2450) loss: 2.302297
(Iteration 271 / 2450) loss: 2.301479
(Iteration 281 / 2450) loss: 2.302853
(Iteration 291 / 2450) loss: 2.302276
(Iteration 301 / 2450) loss: 2.302775
(Iteration 311 / 2450) loss: 2.302713
(Iteration 321 / 2450) loss: 2.302272
(Iteration 331 / 2450) loss: 2.302718
(Iteration 341 / 2450) loss: 2.302253
(Iteration 351 / 2450) loss: 2.302158
(Iteration 361 / 2450) loss: 2.302581
(Iteration 371 / 2450) loss: 2.303429
(Iteration 381 / 2450) loss: 2.302051
(Iteration 391 / 2450) loss: 2.302121
(Iteration 401 / 2450) loss: 2.302341
(Iteration 411 / 2450) loss: 2.300897
(Iteration 421 / 2450) loss: 2.301850
(Iteration 431 / 2450) loss: 2.302428
(Iteration 441 / 2450) loss: 2.303805
```

```
(Iteration 451 / 2450) loss: 2.300806
(Iteration 461 / 2450) loss: 2.302430
(Iteration 471 / 2450) loss: 2.303878
(Iteration 481 / 2450) loss: 2.301761
(Epoch 2 / 10) train acc: 0.108000; val_acc: 0.102000
(Iteration 491 / 2450) loss: 2.302016
(Iteration 501 / 2450) loss: 2.302354
(Iteration 511 / 2450) loss: 2.302019
(Iteration 521 / 2450) loss: 2.301719
(Iteration 531 / 2450) loss: 2.301812
(Iteration 541 / 2450) loss: 2.302270
(Iteration 551 / 2450) loss: 2.304679
(Iteration 561 / 2450) loss: 2.301497
(Iteration 571 / 2450) loss: 2.301157
(Iteration 581 / 2450) loss: 2.303212
(Iteration 591 / 2450) loss: 2.303304
(Iteration 601 / 2450) loss: 2.302809
(Iteration 611 / 2450) loss: 2.304807
(Iteration 621 / 2450) loss: 2.303026
(Iteration 631 / 2450) loss: 2.303173
(Iteration 641 / 2450) loss: 2.302257
(Iteration 651 / 2450) loss: 2.302577
(Iteration 661 / 2450) loss: 2.303261
(Iteration 671 / 2450) loss: 2.302668
(Iteration 681 / 2450) loss: 2.302723
(Iteration 691 / 2450) loss: 2.303279
(Iteration 701 / 2450) loss: 2.302097
(Iteration 711 / 2450) loss: 2.302280
(Iteration 721 / 2450) loss: 2.302645
(Iteration 731 / 2450) loss: 2.302632
(Epoch 3 / 10) train acc: 0.102000; val_acc: 0.102000
(Iteration 741 / 2450) loss: 2.303353
(Iteration 751 / 2450) loss: 2.302601
(Iteration 761 / 2450) loss: 2.302994
(Iteration 771 / 2450) loss: 2.302393
(Iteration 781 / 2450) loss: 2.303067
(Iteration 791 / 2450) loss: 2.302268
(Iteration 801 / 2450) loss: 2.302538
(Iteration 811 / 2450) loss: 2.302557
(Iteration 821 / 2450) loss: 2.302381
(Iteration 831 / 2450) loss: 2.302497
(Iteration 841 / 2450) loss: 2.303153
(Iteration 851 / 2450) loss: 2.302493
(Iteration 861 / 2450) loss: 2.301902
(Iteration 871 / 2450) loss: 2.301923
(Iteration 881 / 2450) loss: 2.301653
(Iteration 891 / 2450) loss: 2.301945
(Iteration 901 / 2450) loss: 2.302282
(Iteration 911 / 2450) loss: 2.302496
(Iteration 921 / 2450) loss: 2.302141
(Iteration 931 / 2450) loss: 2.302618
(Iteration 941 / 2450) loss: 2.302787
(Iteration 951 / 2450) loss: 2.302870
(Iteration 961 / 2450) loss: 2.303062
(Iteration 971 / 2450) loss: 2.302978
(Epoch 4 / 10) train acc: 0.091000; val_acc: 0.098000
(Iteration 981 / 2450) loss: 2.302436
(Iteration 991 / 2450) loss: 2.302892
(Iteration 1001 / 2450) loss: 2.302134
(Iteration 1011 / 2450) loss: 2.301868
(Iteration 1021 / 2450) loss: 2.302695
(Iteration 1031 / 2450) loss: 2.302470
(Iteration 1041 / 2450) loss: 2.302827
(Iteration 1051 / 2450) loss: 2.302544
(Iteration 1061 / 2450) loss: 2.303457
(Iteration 1071 / 2450) loss: 2.303287
(Iteration 1081 / 2450) loss: 2.301591
(Iteration 1091 / 2450) loss: 2.303607
(Iteration 1101 / 2450) loss: 2.302761
```

```
(Iteration 1111 / 2450) loss: 2.303470
(Iteration 1121 / 2450) loss: 2.303335
(Iteration 1131 / 2450) loss: 2.301007
(Iteration 1141 / 2450) loss: 2.302742
(Iteration 1151 / 2450) loss: 2.303987
(Iteration 1161 / 2450) loss: 2.302338
(Iteration 1171 / 2450) loss: 2.302502
(Iteration 1181 / 2450) loss: 2.303385
(Iteration 1191 / 2450) loss: 2.302277
(Iteration 1201 / 2450) loss: 2.302415
(Iteration 1211 / 2450) loss: 2.303023
(Iteration 1221 / 2450) loss: 2.302076
(Epoch 5 / 10) train acc: 0.098000; val_acc: 0.113000
(Iteration 1231 / 2450) loss: 2.302180
(Iteration 1241 / 2450) loss: 2.303232
(Iteration 1251 / 2450) loss: 2.303895
(Iteration 1261 / 2450) loss: 2.301794
(Iteration 1271 / 2450) loss: 2.302117
(Iteration 1281 / 2450) loss: 2.304295
(Iteration 1291 / 2450) loss: 2.302809
(Iteration 1301 / 2450) loss: 2.301791
(Iteration 1311 / 2450) loss: 2.303015
(Iteration 1321 / 2450) loss: 2.302681
(Iteration 1331 / 2450) loss: 2.302032
(Iteration 1341 / 2450) loss: 2.302984
(Iteration 1351 / 2450) loss: 2.302883
(Iteration 1361 / 2450) loss: 2.302700
(Iteration 1371 / 2450) loss: 2.302702
(Iteration 1381 / 2450) loss: 2.302390
(Iteration 1391 / 2450) loss: 2.302406
(Iteration 1401 / 2450) loss: 2.302030
(Iteration 1411 / 2450) loss: 2.302313
(Iteration 1421 / 2450) loss: 2.301653
(Iteration 1431 / 2450) loss: 2.301833
(Iteration 1441 / 2450) loss: 2.301848
(Iteration 1451 / 2450) loss: 2.302444
(Iteration 1461 / 2450) loss: 2.301876
(Epoch 6 / 10) train acc: 0.120000; val_acc: 0.105000
(Iteration 1471 / 2450) loss: 2.302114
(Iteration 1481 / 2450) loss: 2.302244
(Iteration 1491 / 2450) loss: 2.303183
(Iteration 1501 / 2450) loss: 2.303060
(Iteration 1511 / 2450) loss: 2.303265
(Iteration 1521 / 2450) loss: 2.303128
(Iteration 1531 / 2450) loss: 2.303315
(Iteration 1541 / 2450) loss: 2.302651
(Iteration 1551 / 2450) loss: 2.302316
(Iteration 1561 / 2450) loss: 2.302485
(Iteration 1571 / 2450) loss: 2.302721
(Iteration 1581 / 2450) loss: 2.302825
(Iteration 1591 / 2450) loss: 2.302964
(Iteration 1601 / 2450) loss: 2.301624
(Iteration 1611 / 2450) loss: 2.302967
(Iteration 1621 / 2450) loss: 2.303106
(Iteration 1631 / 2450) loss: 2.302820
(Iteration 1641 / 2450) loss: 2.302088
(Iteration 1651 / 2450) loss: 2.303388
(Iteration 1661 / 2450) loss: 2.302603
(Iteration 1671 / 2450) loss: 2.302915
(Iteration 1681 / 2450) loss: 2.302739
(Iteration 1691 / 2450) loss: 2.302582
(Iteration 1701 / 2450) loss: 2.302312
(Iteration 1711 / 2450) loss: 2.302731
(Epoch 7 / 10) train acc: 0.094000; val_acc: 0.078000
(Iteration 1721 / 2450) loss: 2.303285
(Iteration 1731 / 2450) loss: 2.303225
(Iteration 1741 / 2450) loss: 2.302200
(Iteration 1751 / 2450) loss: 2.304349
(Iteration 1761 / 2450) loss: 2.302307
```

```
(Iteration 1771 / 2450) loss: 2.303553
(Iteration 1781 / 2450) loss: 2.302134
(Iteration 1791 / 2450) loss: 2.302256
(Iteration 1801 / 2450) loss: 2.301468
(Iteration 1811 / 2450) loss: 2.302870
(Iteration 1821 / 2450) loss: 2.302684
(Iteration 1831 / 2450) loss: 2.303839
(Iteration 1841 / 2450) loss: 2.302794
(Iteration 1851 / 2450) loss: 2.301458
(Iteration 1861 / 2450) loss: 2.303418
(Iteration 1871 / 2450) loss: 2.302106
(Iteration 1881 / 2450) loss: 2.301435
(Iteration 1891 / 2450) loss: 2.302974
(Iteration 1901 / 2450) loss: 2.301862
(Iteration 1911 / 2450) loss: 2.302231
(Iteration 1921 / 2450) loss: 2.303634
(Iteration 1931 / 2450) loss: 2.303148
(Iteration 1941 / 2450) loss: 2.302379
(Iteration 1951 / 2450) loss: 2.302083
(Epoch 8 / 10) train acc: 0.097000; val_acc: 0.078000
(Iteration 1961 / 2450) loss: 2.302991
(Iteration 1971 / 2450) loss: 2.302823
(Iteration 1981 / 2450) loss: 2.302102
(Iteration 1991 / 2450) loss: 2.303098
(Iteration 2001 / 2450) loss: 2.303840
(Iteration 2011 / 2450) loss: 2.303097
(Iteration 2021 / 2450) loss: 2.303344
(Iteration 2031 / 2450) loss: 2.302631
(Iteration 2041 / 2450) loss: 2.302126
(Iteration 2051 / 2450) loss: 2.303265
(Iteration 2061 / 2450) loss: 2.302935
(Iteration 2071 / 2450) loss: 2.302356
(Iteration 2081 / 2450) loss: 2.303929
(Iteration 2091 / 2450) loss: 2.302690
(Iteration 2101 / 2450) loss: 2.304383
(Iteration 2111 / 2450) loss: 2.301492
(Iteration 2121 / 2450) loss: 2.302874
(Iteration 2131 / 2450) loss: 2.301639
(Iteration 2141 / 2450) loss: 2.303114
(Iteration 2151 / 2450) loss: 2.301527
(Iteration 2161 / 2450) loss: 2.303962
(Iteration 2171 / 2450) loss: 2.303991
(Iteration 2181 / 2450) loss: 2.301863
(Iteration 2191 / 2450) loss: 2.303239
(Iteration 2201 / 2450) loss: 2.303329
(Epoch 9 / 10) train acc: 0.109000; val_acc: 0.107000
(Iteration 2211 / 2450) loss: 2.301879
(Iteration 2221 / 2450) loss: 2.302788
(Iteration 2231 / 2450) loss: 2.303367
(Iteration 2241 / 2450) loss: 2.303059
(Iteration 2251 / 2450) loss: 2.302542
(Iteration 2261 / 2450) loss: 2.302948
(Iteration 2271 / 2450) loss: 2.303193
(Iteration 2281 / 2450) loss: 2.303291
(Iteration 2291 / 2450) loss: 2.303263
(Iteration 2301 / 2450) loss: 2.302139
(Iteration 2311 / 2450) loss: 2.301885
(Iteration 2321 / 2450) loss: 2.303387
(Iteration 2331 / 2450) loss: 2.302320
(Iteration 2341 / 2450) loss: 2.302744
(Iteration 2351 / 2450) loss: 2.303592
(Iteration 2361 / 2450) loss: 2.302440
(Iteration 2371 / 2450) loss: 2.302990
(Iteration 2381 / 2450) loss: 2.303335
(Iteration 2391 / 2450) loss: 2.302201
(Iteration 2401 / 2450) loss: 2.303831
(Iteration 2411 / 2450) loss: 2.301675
(Iteration 2421 / 2450) loss: 2.301356
(Iteration 2431 / 2450) loss: 2.302614
```

```
(Iteration 2441 / 2450) loss: 2.302302
(Epoch 10 / 10) train acc: 0.101000; val_acc: 0.079000
(Iteration 1 / 2450) loss: 2.302585
(Epoch 0 / 10) train acc: 0.088000; val_acc: 0.118000
(Iteration 11 / 2450) loss: 2.302567
(Iteration 21 / 2450) loss: 2.302594
(Iteration 31 / 2450) loss: 2.302575
(Iteration 41 / 2450) loss: 2.302496
(Iteration 51 / 2450) loss: 2.302601
(Iteration 61 / 2450) loss: 2.302498
(Iteration 71 / 2450) loss: 2.302596
(Iteration 81 / 2450) loss: 2.302451
(Iteration 91 / 2450) loss: 2.302461
(Iteration 101 / 2450) loss: 2.302413
(Iteration 111 / 2450) loss: 2.302309
(Iteration 121 / 2450) loss: 2.301279
(Iteration 131 / 2450) loss: 2.299573
(Iteration 141 / 2450) loss: 2.287814
(Iteration 151 / 2450) loss: 2.212082
(Iteration 161 / 2450) loss: 2.218180
(Iteration 171 / 2450) loss: 2.155304
(Iteration 181 / 2450) loss: 2.132654
(Iteration 191 / 2450) loss: 2.016588
(Iteration 201 / 2450) loss: 2.050115
(Iteration 211 / 2450) loss: 2.070387
(Iteration 221 / 2450) loss: 2.051948
(Iteration 231 / 2450) loss: 1.977133
(Iteration 241 / 2450) loss: 2.004837
(Epoch 1 / 10) train acc: 0.215000; val_acc: 0.224000
(Iteration 251 / 2450) loss: 2.032171
(Iteration 261 / 2450) loss: 1.890990
(Iteration 271 / 2450) loss: 1.956629
(Iteration 281 / 2450) loss: 1.953205
(Iteration 291 / 2450) loss: 2.030784
(Iteration 301 / 2450) loss: 1.922183
(Iteration 311 / 2450) loss: 1.980900
(Iteration 321 / 2450) loss: 1.998545
(Iteration 331 / 2450) loss: 1.945991
(Iteration 341 / 2450) loss: 1.856000
(Iteration 351 / 2450) loss: 1.943203
(Iteration 361 / 2450) loss: 1.968202
(Iteration 371 / 2450) loss: 1.945433
(Iteration 381 / 2450) loss: 1.926400
(Iteration 391 / 2450) loss: 1.889539
(Iteration 401 / 2450) loss: 1.827488
(Iteration 411 / 2450) loss: 1.928105
(Iteration 421 / 2450) loss: 1.930909
(Iteration 431 / 2450) loss: 1.891004
(Iteration 441 / 2450) loss: 1.805422
(Iteration 451 / 2450) loss: 1.959240
(Iteration 461 / 2450) loss: 1.893620
(Iteration 471 / 2450) loss: 1.858522
(Iteration 481 / 2450) loss: 1.847942
(Epoch 2 / 10) train acc: 0.284000; val_acc: 0.294000
(Iteration 491 / 2450) loss: 1.738899
(Iteration 501 / 2450) loss: 1.868887
(Iteration 511 / 2450) loss: 1.819701
(Iteration 521 / 2450) loss: 1.779015
(Iteration 531 / 2450) loss: 1.879265
(Iteration 541 / 2450) loss: 1.833871
(Iteration 551 / 2450) loss: 1.894646
(Iteration 561 / 2450) loss: 1.811334
(Iteration 571 / 2450) loss: 1.846911
(Iteration 581 / 2450) loss: 1.826725
(Iteration 591 / 2450) loss: 1.836716
(Iteration 601 / 2450) loss: 1.768888
(Iteration 611 / 2450) loss: 1.731133
(Iteration 621 / 2450) loss: 1.781194
(Iteration 631 / 2450) loss: 1.780639
```

```
(Iteration 641 / 2450) loss: 1.808293
(Iteration 651 / 2450) loss: 1.742513
(Iteration 661 / 2450) loss: 1.741367
(Iteration 671 / 2450) loss: 1.777464
(Iteration 681 / 2450) loss: 1.793763
(Iteration 691 / 2450) loss: 1.873670
(Iteration 701 / 2450) loss: 1.803439
(Iteration 711 / 2450) loss: 1.842007
(Iteration 721 / 2450) loss: 1.841612
(Iteration 731 / 2450) loss: 1.664946
(Epoch 3 / 10) train acc: 0.336000; val_acc: 0.353000
(Iteration 741 / 2450) loss: 1.825934
(Iteration 751 / 2450) loss: 1.736790
(Iteration 761 / 2450) loss: 1.758650
(Iteration 771 / 2450) loss: 1.747881
(Iteration 781 / 2450) loss: 1.795545
(Iteration 791 / 2450) loss: 1.768010
(Iteration 801 / 2450) loss: 1.825513
(Iteration 811 / 2450) loss: 1.757402
(Iteration 821 / 2450) loss: 1.776761
(Iteration 831 / 2450) loss: 1.632701
(Iteration 841 / 2450) loss: 1.825336
(Iteration 851 / 2450) loss: 1.695518
(Iteration 861 / 2450) loss: 1.538261
(Iteration 871 / 2450) loss: 1.736873
(Iteration 881 / 2450) loss: 1.771129
(Iteration 891 / 2450) loss: 1.711023
(Iteration 901 / 2450) loss: 1.659128
(Iteration 911 / 2450) loss: 1.693447
(Iteration 921 / 2450) loss: 1.701044
(Iteration 931 / 2450) loss: 1.755446
(Iteration 941 / 2450) loss: 1.711982
(Iteration 951 / 2450) loss: 1.694379
(Iteration 961 / 2450) loss: 1.703588
(Iteration 971 / 2450) loss: 1.630295
(Epoch 4 / 10) train acc: 0.383000; val_acc: 0.372000
(Iteration 981 / 2450) loss: 1.668824
(Iteration 991 / 2450) loss: 1.716476
(Iteration 1001 / 2450) loss: 1.755742
(Iteration 1011 / 2450) loss: 1.771187
(Iteration 1021 / 2450) loss: 1.725291
(Iteration 1031 / 2450) loss: 1.669099
(Iteration 1041 / 2450) loss: 1.652889
(Iteration 1051 / 2450) loss: 1.626387
(Iteration 1061 / 2450) loss: 1.688984
(Iteration 1071 / 2450) loss: 1.668187
(Iteration 1081 / 2450) loss: 1.614494
(Iteration 1091 / 2450) loss: 1.701104
(Iteration 1101 / 2450) loss: 1.675110
(Iteration 1111 / 2450) loss: 1.695607
(Iteration 1121 / 2450) loss: 1.624820
(Iteration 1131 / 2450) loss: 1.730264
(Iteration 1141 / 2450) loss: 1.680993
(Iteration 1151 / 2450) loss: 1.699815
(Iteration 1161 / 2450) loss: 1.597356
(Iteration 1171 / 2450) loss: 1.637973
(Iteration 1181 / 2450) loss: 1.628738
(Iteration 1191 / 2450) loss: 1.606762
(Iteration 1201 / 2450) loss: 1.603161
(Iteration 1211 / 2450) loss: 1.627777
(Iteration 1221 / 2450) loss: 1.641824
(Epoch 5 / 10) train acc: 0.391000; val_acc: 0.363000
(Iteration 1231 / 2450) loss: 1.728218
(Iteration 1241 / 2450) loss: 1.584011
(Iteration 1251 / 2450) loss: 1.613465
(Iteration 1261 / 2450) loss: 1.607145
(Iteration 1271 / 2450) loss: 1.626678
(Iteration 1281 / 2450) loss: 1.763051
(Iteration 1291 / 2450) loss: 1.715437
```

```
(Iteration 1301 / 2450) loss: 1.608510
(Iteration 1311 / 2450) loss: 1.701179
(Iteration 1321 / 2450) loss: 1.537246
(Iteration 1331 / 2450) loss: 1.648463
(Iteration 1341 / 2450) loss: 1.603930
(Iteration 1351 / 2450) loss: 1.692208
(Iteration 1361 / 2450) loss: 1.612956
(Iteration 1371 / 2450) loss: 1.554488
(Iteration 1381 / 2450) loss: 1.560426
(Iteration 1391 / 2450) loss: 1.731377
(Iteration 1401 / 2450) loss: 1.722085
(Iteration 1411 / 2450) loss: 1.606865
(Iteration 1421 / 2450) loss: 1.629309
(Iteration 1431 / 2450) loss: 1.587569
(Iteration 1441 / 2450) loss: 1.665224
(Iteration 1451 / 2450) loss: 1.564219
(Iteration 1461 / 2450) loss: 1.618877
(Epoch 6 / 10) train acc: 0.371000; val_acc: 0.405000
(Iteration 1471 / 2450) loss: 1.605097
(Iteration 1481 / 2450) loss: 1.564401
(Iteration 1491 / 2450) loss: 1.618261
(Iteration 1501 / 2450) loss: 1.563496
(Iteration 1511 / 2450) loss: 1.504643
(Iteration 1521 / 2450) loss: 1.608166
(Iteration 1531 / 2450) loss: 1.560269
(Iteration 1541 / 2450) loss: 1.515378
(Iteration 1551 / 2450) loss: 1.613033
(Iteration 1561 / 2450) loss: 1.519326
(Iteration 1571 / 2450) loss: 1.664850
(Iteration 1581 / 2450) loss: 1.438713
(Iteration 1591 / 2450) loss: 1.635132
(Iteration 1601 / 2450) loss: 1.526476
(Iteration 1611 / 2450) loss: 1.528175
(Iteration 1621 / 2450) loss: 1.476253
(Iteration 1631 / 2450) loss: 1.544304
(Iteration 1641 / 2450) loss: 1.525558
(Iteration 1651 / 2450) loss: 1.504448
(Iteration 1661 / 2450) loss: 1.550166
(Iteration 1671 / 2450) loss: 1.722611
(Iteration 1681 / 2450) loss: 1.709794
(Iteration 1691 / 2450) loss: 1.535977
(Iteration 1701 / 2450) loss: 1.460203
(Iteration 1711 / 2450) loss: 1.573455
(Epoch 7 / 10) train acc: 0.430000; val_acc: 0.423000
(Iteration 1721 / 2450) loss: 1.553806
(Iteration 1731 / 2450) loss: 1.568875
(Iteration 1741 / 2450) loss: 1.533479
(Iteration 1751 / 2450) loss: 1.602035
(Iteration 1761 / 2450) loss: 1.498806
(Iteration 1771 / 2450) loss: 1.573014
(Iteration 1781 / 2450) loss: 1.651158
(Iteration 1791 / 2450) loss: 1.585230
(Iteration 1801 / 2450) loss: 1.529637
(Iteration 1811 / 2450) loss: 1.488154
(Iteration 1821 / 2450) loss: 1.550350
(Iteration 1831 / 2450) loss: 1.564423
(Iteration 1841 / 2450) loss: 1.480360
(Iteration 1851 / 2450) loss: 1.500280
(Iteration 1861 / 2450) loss: 1.438009
(Iteration 1871 / 2450) loss: 1.567045
(Iteration 1881 / 2450) loss: 1.563679
(Iteration 1891 / 2450) loss: 1.498096
(Iteration 1901 / 2450) loss: 1.499106
(Iteration 1911 / 2450) loss: 1.427890
(Iteration 1921 / 2450) loss: 1.503127
(Iteration 1931 / 2450) loss: 1.620233
(Iteration 1941 / 2450) loss: 1.525680
(Iteration 1951 / 2450) loss: 1.486975
(Epoch 8 / 10) train acc: 0.432000; val_acc: 0.445000
```

```
(Iteration 1961 / 2450) loss: 1.481764
(Iteration 1971 / 2450) loss: 1.425571
(Iteration 1981 / 2450) loss: 1.640827
(Iteration 1991 / 2450) loss: 1.397426
(Iteration 2001 / 2450) loss: 1.325388
(Iteration 2011 / 2450) loss: 1.655935
(Iteration 2021 / 2450) loss: 1.509969
(Iteration 2031 / 2450) loss: 1.493151
(Iteration 2041 / 2450) loss: 1.512691
(Iteration 2051 / 2450) loss: 1.363192
(Iteration 2061 / 2450) loss: 1.403291
(Iteration 2071 / 2450) loss: 1.496805
(Iteration 2081 / 2450) loss: 1.507161
(Iteration 2091 / 2450) loss: 1.332015
(Iteration 2101 / 2450) loss: 1.476896
(Iteration 2111 / 2450) loss: 1.461122
(Iteration 2121 / 2450) loss: 1.463245
(Iteration 2131 / 2450) loss: 1.429415
(Iteration 2141 / 2450) loss: 1.373951
(Iteration 2151 / 2450) loss: 1.309887
(Iteration 2161 / 2450) loss: 1.313249
(Iteration 2171 / 2450) loss: 1.353731
(Iteration 2181 / 2450) loss: 1.476655
(Iteration 2191 / 2450) loss: 1.363772
(Iteration 2201 / 2450) loss: 1.463424
(Epoch 9 / 10) train acc: 0.459000; val_acc: 0.452000
(Iteration 2211 / 2450) loss: 1.492881
(Iteration 2221 / 2450) loss: 1.419268
(Iteration 2231 / 2450) loss: 1.386647
(Iteration 2241 / 2450) loss: 1.568299
(Iteration 2251 / 2450) loss: 1.363573
(Iteration 2261 / 2450) loss: 1.453902
(Iteration 2271 / 2450) loss: 1.410634
(Iteration 2281 / 2450) loss: 1.463311
(Iteration 2291 / 2450) loss: 1.466385
(Iteration 2301 / 2450) loss: 1.461052
(Iteration 2311 / 2450) loss: 1.470093
(Iteration 2321 / 2450) loss: 1.352769
(Iteration 2331 / 2450) loss: 1.463920
(Iteration 2341 / 2450) loss: 1.307802
(Iteration 2351 / 2450) loss: 1.414033
(Iteration 2361 / 2450) loss: 1.435128
(Iteration 2371 / 2450) loss: 1.395289
(Iteration 2381 / 2450) loss: 1.488059
(Iteration 2391 / 2450) loss: 1.419349
(Iteration 2401 / 2450) loss: 1.517804
(Iteration 2411 / 2450) loss: 1.328730
(Iteration 2421 / 2450) loss: 1.382653
(Iteration 2431 / 2450) loss: 1.414015
(Iteration 2441 / 2450) loss: 1.294333
(Epoch 10 / 10) train acc: 0.483000; val_acc: 0.456000
(Iteration 1 / 2450) loss: 3.311104
(Epoch 0 / 10) train acc: 0.098000; val_acc: 0.119000
(Iteration 11 / 2450) loss: 2.278628
(Iteration 21 / 2450) loss: 2.245409
(Iteration 31 / 2450) loss: 2.127497
(Iteration 41 / 2450) loss: 2.035250
(Iteration 51 / 2450) loss: 2.022734
(Iteration 61 / 2450) loss: 1.987338
(Iteration 71 / 2450) loss: 1.984899
(Iteration 81 / 2450) loss: 1.998404
(Iteration 91 / 2450) loss: 1.882989
(Iteration 101 / 2450) loss: 1.868513
(Iteration 111 / 2450) loss: 1.883595
(Iteration 121 / 2450) loss: 1.851517
(Iteration 131 / 2450) loss: 1.744159
(Iteration 141 / 2450) loss: 1.911141
(Iteration 151 / 2450) loss: 1.769741
(Iteration 161 / 2450) loss: 1.782287
```

```
(Iteration 171 / 2450) loss: 1.843224
(Iteration 181 / 2450) loss: 1.902924
(Iteration 191 / 2450) loss: 1.720280
(Iteration 201 / 2450) loss: 1.808634
(Iteration 211 / 2450) loss: 1.719729
(Iteration 221 / 2450) loss: 1.691843
(Iteration 231 / 2450) loss: 1.678982
(Iteration 241 / 2450) loss: 1.688095
(Epoch 1 / 10) train acc: 0.380000; val_acc: 0.392000
(Iteration 251 / 2450) loss: 1.590904
(Iteration 261 / 2450) loss: 1.754898
(Iteration 271 / 2450) loss: 1.701372
(Iteration 281 / 2450) loss: 1.640957
(Iteration 291 / 2450) loss: 1.677886
(Iteration 301 / 2450) loss: 1.639205
(Iteration 311 / 2450) loss: 1.688112
(Iteration 321 / 2450) loss: 1.722419
(Iteration 331 / 2450) loss: 1.579594
(Iteration 341 / 2450) loss: 1.598405
(Iteration 351 / 2450) loss: 1.697102
(Iteration 361 / 2450) loss: 1.650922
(Iteration 371 / 2450) loss: 1.675090
(Iteration 381 / 2450) loss: 1.597011
(Iteration 391 / 2450) loss: 1.616766
(Iteration 401 / 2450) loss: 1.480913
(Iteration 411 / 2450) loss: 1.684320
(Iteration 421 / 2450) loss: 1.498143
(Iteration 431 / 2450) loss: 1.620192
(Iteration 441 / 2450) loss: 1.559871
(Iteration 451 / 2450) loss: 1.610403
(Iteration 461 / 2450) loss: 1.546071
(Iteration 471 / 2450) loss: 1.606757
(Iteration 481 / 2450) loss: 1.485934
(Epoch 2 / 10) train acc: 0.431000; val_acc: 0.444000
(Iteration 491 / 2450) loss: 1.625318
(Iteration 501 / 2450) loss: 1.683358
(Iteration 511 / 2450) loss: 1.622254
(Iteration 521 / 2450) loss: 1.597543
(Iteration 531 / 2450) loss: 1.659898
(Iteration 541 / 2450) loss: 1.583442
(Iteration 551 / 2450) loss: 1.553398
(Iteration 561 / 2450) loss: 1.532634
(Iteration 571 / 2450) loss: 1.547304
(Iteration 581 / 2450) loss: 1.572096
(Iteration 591 / 2450) loss: 1.522864
(Iteration 601 / 2450) loss: 1.585559
(Iteration 611 / 2450) loss: 1.649589
(Iteration 621 / 2450) loss: 1.664729
(Iteration 631 / 2450) loss: 1.470849
(Iteration 641 / 2450) loss: 1.605947
(Iteration 651 / 2450) loss: 1.464483
(Iteration 661 / 2450) loss: 1.509270
(Iteration 671 / 2450) loss: 1.644960
(Iteration 681 / 2450) loss: 1.640079
(Iteration 691 / 2450) loss: 1.548819
(Iteration 701 / 2450) loss: 1.654235
(Iteration 711 / 2450) loss: 1.568710
(Iteration 721 / 2450) loss: 1.513943
(Iteration 731 / 2450) loss: 1.522027
(Epoch 3 / 10) train acc: 0.460000; val_acc: 0.469000
(Iteration 741 / 2450) loss: 1.484952
(Iteration 751 / 2450) loss: 1.504544
(Iteration 761 / 2450) loss: 1.528679
(Iteration 771 / 2450) loss: 1.492364
(Iteration 781 / 2450) loss: 1.473630
(Iteration 791 / 2450) loss: 1.483378
(Iteration 801 / 2450) loss: 1.485220
(Iteration 811 / 2450) loss: 1.380370
(Iteration 821 / 2450) loss: 1.345384
```

```
(Iteration 831 / 2450) loss: 1.511097
(Iteration 841 / 2450) loss: 1.433050
(Iteration 851 / 2450) loss: 1.450613
(Iteration 861 / 2450) loss: 1.401202
(Iteration 871 / 2450) loss: 1.562406
(Iteration 881 / 2450) loss: 1.487168
(Iteration 891 / 2450) loss: 1.386985
(Iteration 901 / 2450) loss: 1.406132
(Iteration 911 / 2450) loss: 1.447474
(Iteration 921 / 2450) loss: 1.478293
(Iteration 931 / 2450) loss: 1.378684
(Iteration 941 / 2450) loss: 1.438876
(Iteration 951 / 2450) loss: 1.448474
(Iteration 961 / 2450) loss: 1.520146
(Iteration 971 / 2450) loss: 1.429314
(Epoch 4 / 10) train acc: 0.449000; val_acc: 0.466000
(Iteration 981 / 2450) loss: 1.363416
(Iteration 991 / 2450) loss: 1.570905
(Iteration 1001 / 2450) loss: 1.504189
(Iteration 1011 / 2450) loss: 1.442563
(Iteration 1021 / 2450) loss: 1.503755
(Iteration 1031 / 2450) loss: 1.540280
(Iteration 1041 / 2450) loss: 1.482653
(Iteration 1051 / 2450) loss: 1.351117
(Iteration 1061 / 2450) loss: 1.476949
(Iteration 1071 / 2450) loss: 1.469325
(Iteration 1081 / 2450) loss: 1.408810
(Iteration 1091 / 2450) loss: 1.361398
(Iteration 1101 / 2450) loss: 1.434017
(Iteration 1111 / 2450) loss: 1.412572
(Iteration 1121 / 2450) loss: 1.433130
(Iteration 1131 / 2450) loss: 1.350557
(Iteration 1141 / 2450) loss: 1.474676
(Iteration 1151 / 2450) loss: 1.483032
(Iteration 1161 / 2450) loss: 1.471633
(Iteration 1171 / 2450) loss: 1.466972
(Iteration 1181 / 2450) loss: 1.409812
(Iteration 1191 / 2450) loss: 1.487978
(Iteration 1201 / 2450) loss: 1.262757
(Iteration 1211 / 2450) loss: 1.482779
(Iteration 1221 / 2450) loss: 1.403511
(Epoch 5 / 10) train acc: 0.516000; val_acc: 0.485000
(Iteration 1231 / 2450) loss: 1.384067
(Iteration 1241 / 2450) loss: 1.302997
(Iteration 1251 / 2450) loss: 1.372578
(Iteration 1261 / 2450) loss: 1.449412
(Iteration 1271 / 2450) loss: 1.452475
(Iteration 1281 / 2450) loss: 1.411433
(Iteration 1291 / 2450) loss: 1.426347
(Iteration 1301 / 2450) loss: 1.407469
(Iteration 1311 / 2450) loss: 1.475345
(Iteration 1321 / 2450) loss: 1.451352
(Iteration 1331 / 2450) loss: 1.462397
(Iteration 1341 / 2450) loss: 1.456237
(Iteration 1351 / 2450) loss: 1.378643
(Iteration 1361 / 2450) loss: 1.444410
(Iteration 1371 / 2450) loss: 1.399892
(Iteration 1381 / 2450) loss: 1.416857
(Iteration 1391 / 2450) loss: 1.268314
(Iteration 1401 / 2450) loss: 1.325935
(Iteration 1411 / 2450) loss: 1.275032
(Iteration 1421 / 2450) loss: 1.374027
(Iteration 1431 / 2450) loss: 1.189318
(Iteration 1441 / 2450) loss: 1.402673
(Iteration 1451 / 2450) loss: 1.341271
(Iteration 1461 / 2450) loss: 1.423126
(Epoch 6 / 10) train acc: 0.494000; val_acc: 0.476000
(Iteration 1471 / 2450) loss: 1.207257
(Iteration 1481 / 2450) loss: 1.368171
```

```
(Iteration 1491 / 2450) loss: 1.316174
(Iteration 1501 / 2450) loss: 1.264633
(Iteration 1511 / 2450) loss: 1.351041
(Iteration 1521 / 2450) loss: 1.246939
(Iteration 1531 / 2450) loss: 1.336497
(Iteration 1541 / 2450) loss: 1.392041
(Iteration 1551 / 2450) loss: 1.341927
(Iteration 1561 / 2450) loss: 1.403698
(Iteration 1571 / 2450) loss: 1.285258
(Iteration 1581 / 2450) loss: 1.466774
(Iteration 1591 / 2450) loss: 1.315066
(Iteration 1601 / 2450) loss: 1.324019
(Iteration 1611 / 2450) loss: 1.348532
(Iteration 1621 / 2450) loss: 1.296901
(Iteration 1631 / 2450) loss: 1.335387
(Iteration 1641 / 2450) loss: 1.347531
(Iteration 1651 / 2450) loss: 1.381540
(Iteration 1661 / 2450) loss: 1.304663
(Iteration 1671 / 2450) loss: 1.349117
(Iteration 1681 / 2450) loss: 1.273462
(Iteration 1691 / 2450) loss: 1.294374
(Iteration 1701 / 2450) loss: 1.281780
(Iteration 1711 / 2450) loss: 1.305652
(Epoch 7 / 10) train acc: 0.537000; val_acc: 0.485000
(Iteration 1721 / 2450) loss: 1.251437
(Iteration 1731 / 2450) loss: 1.386430
(Iteration 1741 / 2450) loss: 1.236674
(Iteration 1751 / 2450) loss: 1.400996
(Iteration 1761 / 2450) loss: 1.269360
(Iteration 1771 / 2450) loss: 1.317914
(Iteration 1781 / 2450) loss: 1.271520
(Iteration 1791 / 2450) loss: 1.219073
(Iteration 1801 / 2450) loss: 1.309109
(Iteration 1811 / 2450) loss: 1.383272
(Iteration 1821 / 2450) loss: 1.334384
(Iteration 1831 / 2450) loss: 1.448770
(Iteration 1841 / 2450) loss: 1.259163
(Iteration 1851 / 2450) loss: 1.373860
(Iteration 1861 / 2450) loss: 1.270917
(Iteration 1871 / 2450) loss: 1.416303
(Iteration 1881 / 2450) loss: 1.200510
(Iteration 1891 / 2450) loss: 1.268229
(Iteration 1901 / 2450) loss: 1.138240
(Iteration 1911 / 2450) loss: 1.336101
(Iteration 1921 / 2450) loss: 1.163212
(Iteration 1931 / 2450) loss: 1.259284
(Iteration 1941 / 2450) loss: 1.349405
(Iteration 1951 / 2450) loss: 1.228352
(Epoch 8 / 10) train acc: 0.532000; val_acc: 0.475000
(Iteration 1961 / 2450) loss: 1.208650
(Iteration 1971 / 2450) loss: 1.268402
(Iteration 1981 / 2450) loss: 1.286433
(Iteration 1991 / 2450) loss: 1.356240
(Iteration 2001 / 2450) loss: 1.302914
(Iteration 2011 / 2450) loss: 1.279458
(Iteration 2021 / 2450) loss: 1.255281
(Iteration 2031 / 2450) loss: 1.393779
(Iteration 2041 / 2450) loss: 1.303403
(Iteration 2051 / 2450) loss: 1.361494
(Iteration 2061 / 2450) loss: 1.239011
(Iteration 2071 / 2450) loss: 1.168616
(Iteration 2081 / 2450) loss: 1.275526
(Iteration 2091 / 2450) loss: 1.301649
(Iteration 2101 / 2450) loss: 1.415241
(Iteration 2111 / 2450) loss: 1.313274
(Iteration 2121 / 2450) loss: 1.207314
(Iteration 2131 / 2450) loss: 1.304803
(Iteration 2141 / 2450) loss: 1.300128
(Iteration 2151 / 2450) loss: 1.282299
```

```
(Iteration 2161 / 2450) loss: 1.213250
(Iteration 2171 / 2450) loss: 1.213866
(Iteration 2181 / 2450) loss: 1.414073
(Iteration 2191 / 2450) loss: 1.333258
(Iteration 2201 / 2450) loss: 1.342450
(Epoch 9 / 10) train acc: 0.541000; val_acc: 0.485000
(Iteration 2211 / 2450) loss: 1.287813
(Iteration 2221 / 2450) loss: 1.247092
(Iteration 2231 / 2450) loss: 1.364235
(Iteration 2241 / 2450) loss: 1.223800
(Iteration 2251 / 2450) loss: 1.318793
(Iteration 2261 / 2450) loss: 1.203106
(Iteration 2271 / 2450) loss: 1.309370
(Iteration 2281 / 2450) loss: 1.219248
(Iteration 2291 / 2450) loss: 1.113056
(Iteration 2301 / 2450) loss: 1.288504
(Iteration 2311 / 2450) loss: 1.249728
(Iteration 2321 / 2450) loss: 1.383503
(Iteration 2331 / 2450) loss: 1.243273
(Iteration 2341 / 2450) loss: 1.161287
(Iteration 2351 / 2450) loss: 1.092580
(Iteration 2361 / 2450) loss: 1.150709
(Iteration 2371 / 2450) loss: 1.128000
(Iteration 2381 / 2450) loss: 1.294059
(Iteration 2391 / 2450) loss: 1.069275
(Iteration 2401 / 2450) loss: 1.233369
(Iteration 2411 / 2450) loss: 1.267699
(Iteration 2421 / 2450) loss: 1.162632
(Iteration 2431 / 2450) loss: 1.254346
(Iteration 2441 / 2450) loss: 1.241198
(Epoch 10 / 10) train acc: 0.596000; val_acc: 0.493000
(Iteration 1 / 2450) loss: 2.302585
(Epoch 0 / 10) train acc: 0.101000; val_acc: 0.102000
(Iteration 11 / 2450) loss: 2.302575
(Iteration 21 / 2450) loss: 2.302597
(Iteration 31 / 2450) loss: 2.302564
(Iteration 41 / 2450) loss: 2.302619
(Iteration 51 / 2450) loss: 2.302527
(Iteration 61 / 2450) loss: 2.302503
(Iteration 71 / 2450) loss: 2.302672
(Iteration 81 / 2450) loss: 2.302656
(Iteration 91 / 2450) loss: 2.302491
(Iteration 101 / 2450) loss: 2.302577
(Iteration 111 / 2450) loss: 2.302549
(Iteration 121 / 2450) loss: 2.302638
(Iteration 131 / 2450) loss: 2.302583
(Iteration 141 / 2450) loss: 2.302591
(Iteration 151 / 2450) loss: 2.302519
(Iteration 161 / 2450) loss: 2.302554
(Iteration 171 / 2450) loss: 2.302416
(Iteration 181 / 2450) loss: 2.302682
(Iteration 191 / 2450) loss: 2.302700
(Iteration 201 / 2450) loss: 2.302819
(Iteration 211 / 2450) loss: 2.302660
(Iteration 221 / 2450) loss: 2.302693
(Iteration 231 / 2450) loss: 2.302648
(Iteration 241 / 2450) loss: 2.302502
(Epoch 1 / 10) train acc: 0.096000; val_acc: 0.102000
(Iteration 251 / 2450) loss: 2.302501
(Iteration 261 / 2450) loss: 2.302495
(Iteration 271 / 2450) loss: 2.302566
(Iteration 281 / 2450) loss: 2.302708
(Iteration 291 / 2450) loss: 2.302623
(Iteration 301 / 2450) loss: 2.302920
(Iteration 311 / 2450) loss: 2.302668
(Iteration 321 / 2450) loss: 2.302423
(Iteration 331 / 2450) loss: 2.302486
(Iteration 341 / 2450) loss: 2.302719
(Iteration 351 / 2450) loss: 2.302638
```

```
(Iteration 361 / 2450) loss: 2.302816
(Iteration 371 / 2450) loss: 2.302674
(Iteration 381 / 2450) loss: 2.302578
(Iteration 391 / 2450) loss: 2.302495
(Iteration 401 / 2450) loss: 2.302464
(Iteration 411 / 2450) loss: 2.302673
(Iteration 421 / 2450) loss: 2.302640
(Iteration 431 / 2450) loss: 2.302769
(Iteration 441 / 2450) loss: 2.302520
(Iteration 451 / 2450) loss: 2.302483
(Iteration 461 / 2450) loss: 2.302742
(Iteration 471 / 2450) loss: 2.302830
(Iteration 481 / 2450) loss: 2.302493
(Epoch 2 / 10) train acc: 0.094000; val_acc: 0.078000
(Iteration 491 / 2450) loss: 2.303030
(Iteration 501 / 2450) loss: 2.302553
(Iteration 511 / 2450) loss: 2.302618
(Iteration 521 / 2450) loss: 2.302626
(Iteration 531 / 2450) loss: 2.302589
(Iteration 541 / 2450) loss: 2.302420
(Iteration 551 / 2450) loss: 2.302488
(Iteration 561 / 2450) loss: 2.302631
(Iteration 571 / 2450) loss: 2.302595
(Iteration 581 / 2450) loss: 2.302543
(Iteration 591 / 2450) loss: 2.302572
(Iteration 601 / 2450) loss: 2.302624
(Iteration 611 / 2450) loss: 2.302405
(Iteration 621 / 2450) loss: 2.302605
(Iteration 631 / 2450) loss: 2.302733
(Iteration 641 / 2450) loss: 2.302529
(Iteration 651 / 2450) loss: 2.302514
(Iteration 661 / 2450) loss: 2.302525
(Iteration 671 / 2450) loss: 2.302521
(Iteration 681 / 2450) loss: 2.302510
(Iteration 691 / 2450) loss: 2.302798
(Iteration 701 / 2450) loss: 2.302361
(Iteration 711 / 2450) loss: 2.302559
(Iteration 721 / 2450) loss: 2.302540
(Iteration 731 / 2450) loss: 2.302276
(Epoch 3 / 10) train acc: 0.085000; val_acc: 0.078000
(Iteration 741 / 2450) loss: 2.302555
(Iteration 751 / 2450) loss: 2.302382
(Iteration 761 / 2450) loss: 2.302761
(Iteration 771 / 2450) loss: 2.302670
(Iteration 781 / 2450) loss: 2.302693
(Iteration 791 / 2450) loss: 2.302567
(Iteration 801 / 2450) loss: 2.302591
(Iteration 811 / 2450) loss: 2.302816
(Iteration 821 / 2450) loss: 2.302439
(Iteration 831 / 2450) loss: 2.302142
(Iteration 841 / 2450) loss: 2.302784
(Iteration 851 / 2450) loss: 2.302705
(Iteration 861 / 2450) loss: 2.302567
(Iteration 871 / 2450) loss: 2.302357
(Iteration 881 / 2450) loss: 2.302658
(Iteration 891 / 2450) loss: 2.302751
(Iteration 901 / 2450) loss: 2.302580
(Iteration 911 / 2450) loss: 2.302574
(Iteration 921 / 2450) loss: 2.302656
(Iteration 931 / 2450) loss: 2.302770
(Iteration 941 / 2450) loss: 2.302889
(Iteration 951 / 2450) loss: 2.302609
(Iteration 961 / 2450) loss: 2.302511
(Iteration 971 / 2450) loss: 2.302720
(Epoch 4 / 10) train acc: 0.112000; val_acc: 0.079000
(Iteration 981 / 2450) loss: 2.302486
(Iteration 991 / 2450) loss: 2.302585
(Iteration 1001 / 2450) loss: 2.302422
(Iteration 1011 / 2450) loss: 2.302438
```

```
(Iteration 1021 / 2450) loss: 2.302668
(Iteration 1031 / 2450) loss: 2.302435
(Iteration 1041 / 2450) loss: 2.302383
(Iteration 1051 / 2450) loss: 2.302630
(Iteration 1061 / 2450) loss: 2.302586
(Iteration 1071 / 2450) loss: 2.302477
(Iteration 1081 / 2450) loss: 2.302929
(Iteration 1091 / 2450) loss: 2.302618
(Iteration 1101 / 2450) loss: 2.302958
(Iteration 1111 / 2450) loss: 2.302455
(Iteration 1121 / 2450) loss: 2.302773
(Iteration 1131 / 2450) loss: 2.302396
(Iteration 1141 / 2450) loss: 2.302495
(Iteration 1151 / 2450) loss: 2.302648
(Iteration 1161 / 2450) loss: 2.302539
(Iteration 1171 / 2450) loss: 2.302540
(Iteration 1181 / 2450) loss: 2.302616
(Iteration 1191 / 2450) loss: 2.302408
(Iteration 1201 / 2450) loss: 2.302359
(Iteration 1211 / 2450) loss: 2.302552
(Iteration 1221 / 2450) loss: 2.302439
(Epoch 5 / 10) train acc: 0.090000; val_acc: 0.087000
(Iteration 1231 / 2450) loss: 2.302223
(Iteration 1241 / 2450) loss: 2.303011
(Iteration 1251 / 2450) loss: 2.302656
(Iteration 1261 / 2450) loss: 2.302669
(Iteration 1271 / 2450) loss: 2.302352
(Iteration 1281 / 2450) loss: 2.302722
(Iteration 1291 / 2450) loss: 2.302685
(Iteration 1301 / 2450) loss: 2.302287
(Iteration 1311 / 2450) loss: 2.302636
(Iteration 1321 / 2450) loss: 2.302273
(Iteration 1331 / 2450) loss: 2.302633
(Iteration 1341 / 2450) loss: 2.302296
(Iteration 1351 / 2450) loss: 2.302851
(Iteration 1361 / 2450) loss: 2.302762
(Iteration 1371 / 2450) loss: 2.302448
(Iteration 1381 / 2450) loss: 2.302709
(Iteration 1391 / 2450) loss: 2.302199
(Iteration 1401 / 2450) loss: 2.302593
(Iteration 1411 / 2450) loss: 2.302663
(Iteration 1421 / 2450) loss: 2.302533
(Iteration 1431 / 2450) loss: 2.302605
(Iteration 1441 / 2450) loss: 2.302798
(Iteration 1451 / 2450) loss: 2.302795
(Iteration 1461 / 2450) loss: 2.302464
(Epoch 6 / 10) train acc: 0.097000; val_acc: 0.087000
(Iteration 1471 / 2450) loss: 2.301875
(Iteration 1481 / 2450) loss: 2.302683
(Iteration 1491 / 2450) loss: 2.302481
(Iteration 1501 / 2450) loss: 2.302669
(Iteration 1511 / 2450) loss: 2.302130
(Iteration 1521 / 2450) loss: 2.302769
(Iteration 1531 / 2450) loss: 2.302607
(Iteration 1541 / 2450) loss: 2.302598
(Iteration 1551 / 2450) loss: 2.302646
(Iteration 1561 / 2450) loss: 2.302474
(Iteration 1571 / 2450) loss: 2.302457
(Iteration 1581 / 2450) loss: 2.302555
(Iteration 1591 / 2450) loss: 2.302488
(Iteration 1601 / 2450) loss: 2.302469
(Iteration 1611 / 2450) loss: 2.302532
(Iteration 1621 / 2450) loss: 2.302381
(Iteration 1631 / 2450) loss: 2.302705
(Iteration 1641 / 2450) loss: 2.302362
(Iteration 1651 / 2450) loss: 2.302528
(Iteration 1661 / 2450) loss: 2.302778
(Iteration 1671 / 2450) loss: 2.302438
(Iteration 1681 / 2450) loss: 2.302604
```

```
(Iteration 1691 / 2450) loss: 2.302479
(Iteration 1701 / 2450) loss: 2.301979
(Iteration 1711 / 2450) loss: 2.302679
(Epoch 7 / 10) train acc: 0.120000; val_acc: 0.079000
(Iteration 1721 / 2450) loss: 2.302321
(Iteration 1731 / 2450) loss: 2.302961
(Iteration 1741 / 2450) loss: 2.302743
(Iteration 1751 / 2450) loss: 2.302607
(Iteration 1761 / 2450) loss: 2.302549
(Iteration 1771 / 2450) loss: 2.303052
(Iteration 1781 / 2450) loss: 2.302418
(Iteration 1791 / 2450) loss: 2.302770
(Iteration 1801 / 2450) loss: 2.302498
(Iteration 1811 / 2450) loss: 2.302358
(Iteration 1821 / 2450) loss: 2.302563
(Iteration 1831 / 2450) loss: 2.302386
(Iteration 1841 / 2450) loss: 2.302351
(Iteration 1851 / 2450) loss: 2.302754
(Iteration 1861 / 2450) loss: 2.302662
(Iteration 1871 / 2450) loss: 2.302665
(Iteration 1881 / 2450) loss: 2.302816
(Iteration 1891 / 2450) loss: 2.302595
(Iteration 1901 / 2450) loss: 2.302296
(Iteration 1911 / 2450) loss: 2.302686
(Iteration 1921 / 2450) loss: 2.302674
(Iteration 1931 / 2450) loss: 2.302600
(Iteration 1941 / 2450) loss: 2.302384
(Iteration 1951 / 2450) loss: 2.302486
(Epoch 8 / 10) train acc: 0.112000; val_acc: 0.079000
(Iteration 1961 / 2450) loss: 2.302861
(Iteration 1971 / 2450) loss: 2.302480
(Iteration 1981 / 2450) loss: 2.302476
(Iteration 1991 / 2450) loss: 2.302393
(Iteration 2001 / 2450) loss: 2.302743
(Iteration 2011 / 2450) loss: 2.302710
(Iteration 2021 / 2450) loss: 2.302957
(Iteration 2031 / 2450) loss: 2.302359
(Iteration 2041 / 2450) loss: 2.302525
(Iteration 2051 / 2450) loss: 2.303076
(Iteration 2061 / 2450) loss: 2.302296
(Iteration 2071 / 2450) loss: 2.302747
(Iteration 2081 / 2450) loss: 2.302855
(Iteration 2091 / 2450) loss: 2.302716
(Iteration 2101 / 2450) loss: 2.302419
(Iteration 2111 / 2450) loss: 2.302449
(Iteration 2121 / 2450) loss: 2.302418
(Iteration 2131 / 2450) loss: 2.302645
(Iteration 2141 / 2450) loss: 2.302347
(Iteration 2151 / 2450) loss: 2.302438
(Iteration 2161 / 2450) loss: 2.302700
(Iteration 2171 / 2450) loss: 2.302709
(Iteration 2181 / 2450) loss: 2.302325
(Iteration 2191 / 2450) loss: 2.302785
(Iteration 2201 / 2450) loss: 2.302562
(Epoch 9 / 10) train acc: 0.089000; val_acc: 0.079000
(Iteration 2211 / 2450) loss: 2.302552
(Iteration 2221 / 2450) loss: 2.302477
(Iteration 2231 / 2450) loss: 2.302372
(Iteration 2241 / 2450) loss: 2.302806
(Iteration 2251 / 2450) loss: 2.302567
(Iteration 2261 / 2450) loss: 2.302899
(Iteration 2271 / 2450) loss: 2.302108
(Iteration 2281 / 2450) loss: 2.302668
(Iteration 2291 / 2450) loss: 2.302962
(Iteration 2301 / 2450) loss: 2.302846
(Iteration 2311 / 2450) loss: 2.302618
(Iteration 2321 / 2450) loss: 2.302343
(Iteration 2331 / 2450) loss: 2.302828
(Iteration 2341 / 2450) loss: 2.302782
```

```
(Iteration 2351 / 2450) loss: 2.302402
(Iteration 2361 / 2450) loss: 2.303011
(Iteration 2371 / 2450) loss: 2.302319
(Iteration 2381 / 2450) loss: 2.302758
(Iteration 2391 / 2450) loss: 2.302604
(Iteration 2401 / 2450) loss: 2.302397
(Iteration 2411 / 2450) loss: 2.302575
(Iteration 2421 / 2450) loss: 2.302783
(Iteration 2431 / 2450) loss: 2.302623
(Iteration 2441 / 2450) loss: 2.302690
(Epoch 10 / 10) train acc: 0.117000; val_acc: 0.079000
```

Test Your Model!

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

In [48]:

```
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:  0.532
Test set accuracy:  0.528
```