Dropout

In this notebook, you will implement dropout. Then we will ask you to train a network with batchnorm and dropout, and acheive over 60% accuracy on CIFAR-10.

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

In [1]:

```
## Import and setups

import time
import numpy as np
import matplotlib.pyplot as plt
from nndl.fc_net import *
from nndl.layers import *
from cs231n.data_utils import get_CIFAR10_data
from cs231n.gradient_check import eval_numerical_gradient, eval_numerical_gradient_
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'

# for auto-reloading external modules
# see http://stackoverflow.com/questions/1907993/autoreload-of-modules-in-ipython
%load_ext autoreload
%autoreload 2

def rel_error(x, y):
  """ returns relative error """
  return np.max(np.abs(x - y) / (np.maximum(1e-8, np.abs(x) + np.abs(y))))
```

In [2]:

```
# Load the (preprocessed) CIFAR10 data.

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

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

Dropout forward pass

Implement the training and test time dropout forward pass, `dropout_forward` , in `nndl/layers.py` . After that, test your implementation by running the following cell.

In [3]:

```
x = np.random.randn(500, 500) + 10

for p in [0.3, 0.6, 0.75]:
  out, _ = dropout_forward(x, {'mode': 'train', 'p': p})
  out_test, _ = dropout_forward(x, {'mode': 'test', 'p': p})

  print('Running tests with p = ', p)
  print('Mean of input: ', x.mean())
  print('Mean of train-time output: ', out.mean())
  print('Mean of test-time output: ', out_test.mean())
  print('Fraction of train-time output set to zero: ', (out == 0).mean())
  print('Fraction of test-time output set to zero: ', (out_test == 0).mean())
```

```
Running tests with p =  0.3
Mean of input:  9.999173896404256
Mean of train-time output:  10.03564461966431
Mean of test-time output:  9.999173896404256
Fraction of train-time output set to zero:  0.69882
Fraction of test-time output set to zero:  0.0
Running tests with p =  0.6
Mean of input:  9.999173896404256
Mean of train-time output:  9.981303066021745
Mean of test-time output:  9.999173896404256
Fraction of train-time output set to zero:  0.401032
Fraction of test-time output set to zero:  0.0
Running tests with p =  0.75
Mean of input:  9.999173896404256
Mean of train-time output:  10.001870004386827
Mean of test-time output:  9.999173896404256
Fraction of train-time output set to zero:  0.249708
Fraction of test-time output set to zero:  0.0
```

Dropout backward pass

Implement the backward pass, `dropout_backward` , in `nndl/layers.py` . After that, test your gradients by running the following cell:

```
In [4]:
```

```
x = np.random.randn(10, 10) + 10
dout = np.random.randn(*x.shape)

dropout_param = {'mode': 'train', 'p': 0.8, 'seed': 123}
out, cache = dropout_forward(x, dropout_param)
dx = dropout_backward(dout, cache)
dx_num = eval_numerical_gradient_array(lambda xx: dropout_forward(xx, dropout_param)

print('dx relative error: ', rel_error(dx, dx_num))
```

```
dx relative error:  5.445613574740722e-11
```

Implement a fully connected neural network with dropout layers

Modify the `FullyConnectedNet()` class in `nndl/fc_net.py` to incorporate dropout. A dropout layer should be incorporated after every ReLU layer. Concretely, there shouldn't be a dropout at the output layer since there is no ReLU at the output layer. You will need to modify the class in the following areas:

(1) In the forward pass, you will need to incorporate a dropout layer after every relu layer.

(2) In the backward pass, you will need to incorporate a dropout backward pass layer.

Check your implementation by running the following code. Our W1 gradient relative error is on the order of 1e-6 (the largest of all the relative errors).

Dropout as a regularizer

In class, we claimed that dropout acts as a regularizer by effectively bagging. To check this, we will train two small networks, one with dropout and one without dropout.

In [5]:

```
N, D, H1, H2, C = 2, 15, 20, 30, 10
X = np.random.randn(N, D)
y = np.random.randint(C, size=(N,))

for dropout in [0, 0.25, 0.5]:
  print('Running check with dropout = ', dropout)
  model = FullyConnectedNet([H1, H2], input_dim=D, num_classes=C,
                            weight_scale=5e-2, dtype=np.float64,
                            dropout=dropout, seed=123)

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

  for name in sorted(grads):
    f = lambda _: model.loss(X, y)[0]
    grad_num = eval_numerical_gradient(f, model.params[name], verbose=False, h=1e-5
    print('{} relative error: {}'.format(name, rel_error(grad_num, grads[name])))
  print('\n')
```

```
Running check with dropout =  0
Initial loss:  2.3051938247860218
W1 relative error: 7.711419522175973e-07
W2 relative error: 1.5034484932141387e-05
W3 relative error: 6.398873279808276e-07
b1 relative error: 2.9369574464090924e-06
b2 relative error: 6.320762652162454e-07
b3 relative error: 5.016029374797846e-07


Running check with dropout =  0.25
Initial loss:  2.305429202994099
W1 relative error: 5.024649969461409e-07
W2 relative error: 5.026393983338925e-07
W3 relative error: 5.292126245232686e-07
b1 relative error: 5.029711099712491e-07
b2 relative error: 5.02533626843244e-07
b3 relative error: 5.026138778638627e-07


Running check with dropout =  0.5
Initial loss:  2.307550515805681
W1 relative error: 5.399786757971378e-07
W2 relative error: 5.345642293235742e-07
W3 relative error: 5.440162773705216e-07
b1 relative error: 5.139377993145615e-07
b2 relative error: 5.056039855824707e-07
b3 relative error: 5.060262454954197e-07
```

In [6]:

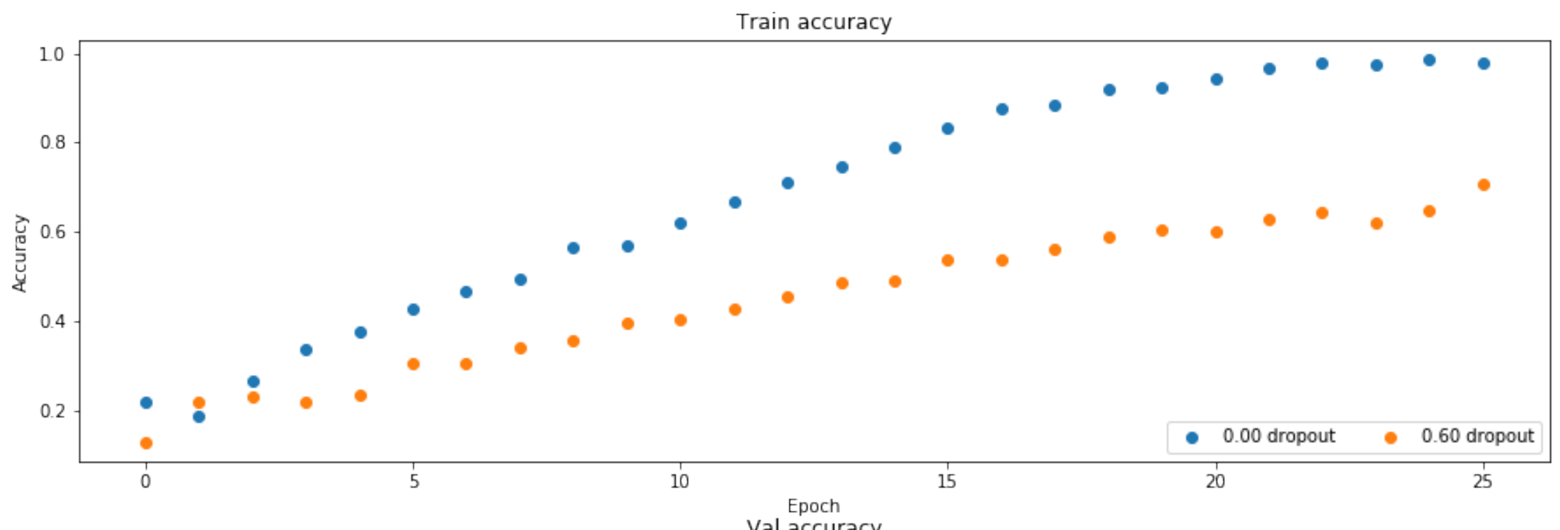
```python
# Train two identical nets, one with dropout and one without

num_train = 500
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 = {}
dropout_choices = [0, 0.6]
for dropout in dropout_choices:
  model = FullyConnectedNet([100, 100, 100], dropout=dropout)

  solver = Solver(model, small_data,
                  num_epochs=25, batch_size=100,
                  update_rule='adam',
                  optim_config={
                    'learning_rate': 5e-4,
                  },
                  verbose=True, print_every=100)
  solver.train()
  solvers[dropout] = solver
```

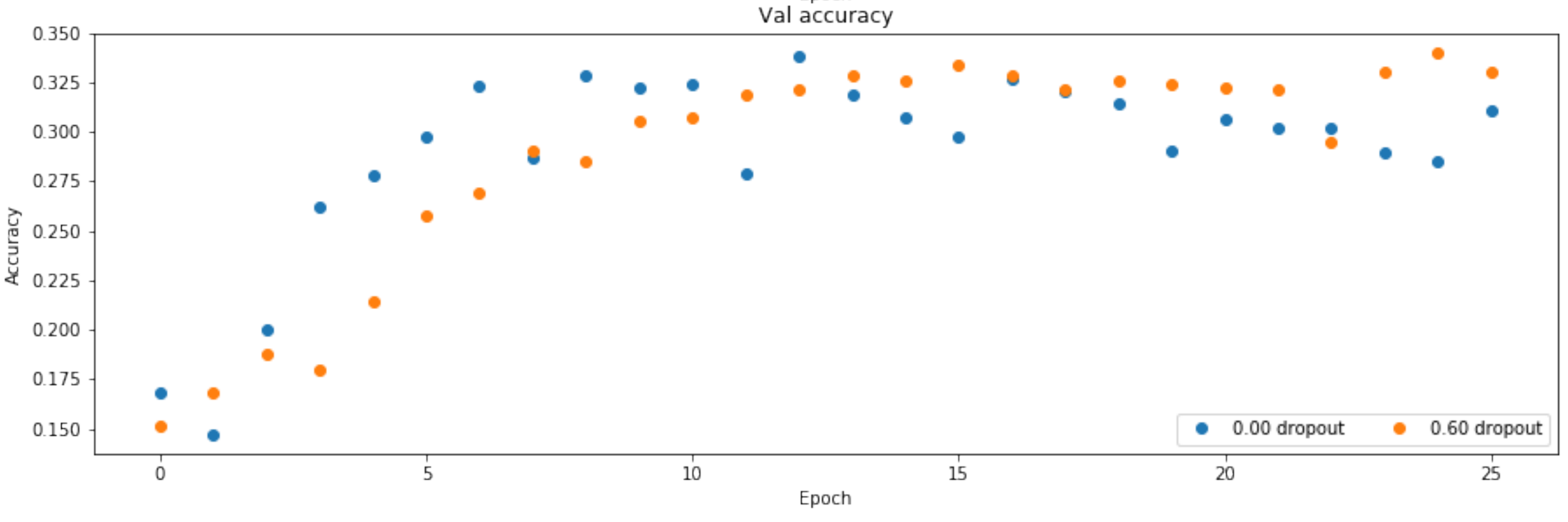
```
(Iteration 1 / 125) loss: 2.300804
(Epoch 0 / 25) train acc: 0.220000; val_acc: 0.168000
(Epoch 1 / 25) train acc: 0.188000; val_acc: 0.147000
(Epoch 2 / 25) train acc: 0.266000; val_acc: 0.200000
(Epoch 3 / 25) train acc: 0.338000; val_acc: 0.262000
(Epoch 4 / 25) train acc: 0.378000; val_acc: 0.278000
(Epoch 5 / 25) train acc: 0.428000; val_acc: 0.297000
(Epoch 6 / 25) train acc: 0.468000; val_acc: 0.323000
(Epoch 7 / 25) train acc: 0.494000; val_acc: 0.287000
(Epoch 8 / 25) train acc: 0.566000; val_acc: 0.328000
(Epoch 9 / 25) train acc: 0.572000; val_acc: 0.322000
(Epoch 10 / 25) train acc: 0.622000; val_acc: 0.324000
(Epoch 11 / 25) train acc: 0.670000; val_acc: 0.279000
(Epoch 12 / 25) train acc: 0.710000; val_acc: 0.338000
(Epoch 13 / 25) train acc: 0.746000; val_acc: 0.319000
(Epoch 14 / 25) train acc: 0.792000; val_acc: 0.307000
(Epoch 15 / 25) train acc: 0.834000; val_acc: 0.297000
(Epoch 16 / 25) train acc: 0.876000; val_acc: 0.327000
(Epoch 17 / 25) train acc: 0.886000; val_acc: 0.320000
(Epoch 18 / 25) train acc: 0.918000; val_acc: 0.314000
(Epoch 19 / 25) train acc: 0.922000; val_acc: 0.290000
(Epoch 20 / 25) train acc: 0.944000; val_acc: 0.306000
(Iteration 101 / 125) loss: 0.156105
(Epoch 21 / 25) train acc: 0.968000; val_acc: 0.302000
(Epoch 22 / 25) train acc: 0.978000; val_acc: 0.302000
(Epoch 23 / 25) train acc: 0.976000; val_acc: 0.289000
(Epoch 24 / 25) train acc: 0.986000; val_acc: 0.285000
(Epoch 25 / 25) train acc: 0.978000; val_acc: 0.311000
```

```
(Iteration 1 / 125) loss: 2.309402
(Epoch 0 / 25) train acc: 0.130000; val_acc: 0.151000
(Epoch 1 / 25) train acc: 0.220000; val_acc: 0.168000
(Epoch 2 / 25) train acc: 0.232000; val_acc: 0.188000
(Epoch 3 / 25) train acc: 0.220000; val_acc: 0.180000
(Epoch 4 / 25) train acc: 0.236000; val_acc: 0.214000
(Epoch 5 / 25) train acc: 0.306000; val_acc: 0.258000
(Epoch 6 / 25) train acc: 0.308000; val_acc: 0.269000
(Epoch 7 / 25) train acc: 0.344000; val_acc: 0.290000
(Epoch 8 / 25) train acc: 0.358000; val_acc: 0.285000
(Epoch 9 / 25) train acc: 0.396000; val_acc: 0.305000
(Epoch 10 / 25) train acc: 0.406000; val_acc: 0.307000
(Epoch 11 / 25) train acc: 0.428000; val_acc: 0.319000
(Epoch 12 / 25) train acc: 0.458000; val_acc: 0.321000
(Epoch 13 / 25) train acc: 0.486000; val_acc: 0.328000
(Epoch 14 / 25) train acc: 0.490000; val_acc: 0.326000
(Epoch 15 / 25) train acc: 0.540000; val_acc: 0.334000
(Epoch 16 / 25) train acc: 0.540000; val_acc: 0.328000
(Epoch 17 / 25) train acc: 0.562000; val_acc: 0.321000
(Epoch 18 / 25) train acc: 0.590000; val_acc: 0.326000
(Epoch 19 / 25) train acc: 0.604000; val_acc: 0.324000
(Epoch 20 / 25) train acc: 0.600000; val_acc: 0.322000
(Iteration 101 / 125) loss: 1.377055
(Epoch 21 / 25) train acc: 0.628000; val_acc: 0.321000
(Epoch 22 / 25) train acc: 0.644000; val_acc: 0.295000
(Epoch 23 / 25) train acc: 0.622000; val_acc: 0.330000
(Epoch 24 / 25) train acc: 0.650000; val_acc: 0.340000
(Epoch 25 / 25) train acc: 0.708000; val_acc: 0.330000
```

In [7]:

```
# Plot train and validation accuracies of the two models

train_accs = []
val_accs = []
for dropout in dropout_choices:
  solver = solvers[dropout]
  train_accs.append(solver.train_acc_history[-1])
  val_accs.append(solver.val_acc_history[-1])

plt.subplot(3, 1, 1)
for dropout in dropout_choices:
  plt.plot(solvers[dropout].train_acc_history, 'o', label='%.2f dropout' % dropout)
plt.title('Train accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(ncol=2, loc='lower right')

plt.subplot(3, 1, 2)
for dropout in dropout_choices:
  plt.plot(solvers[dropout].val_acc_history, 'o', label='%.2f dropout' % dropout)
plt.title('Val accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(ncol=2, loc='lower right')

plt.gcf().set_size_inches(15, 15)
plt.show()
```





Question

Based off the results of this experiment, is dropout performing regularization? Explain your answer.

Answer:

Yes. It can be seen that by using dropout, the training accuracy decreases while the validation accuracy increases. Dropout can help reduce the overfitting problem so that is performing regularization.

Final part of the assignment

Get over 60% validation accuracy on CIFAR-10 by using the layers you have implemented. You will be graded according to the following equation:

min(floor((X - 32%)) / 28%, 1) where if you get 60% or higher validation accuracy, you get full points.

In [8]:

```
# ================================================================ #
# YOUR CODE HERE:
#   Implement a FC-net that achieves at least 60% validation accuracy
#   on CIFAR-10.
# ================================================================ #
data = get_CIFAR10_data()
hidden_dims = [500, 500, 500, 500]
whole_data = {
  'X_train': data['X_train'],
  'y_train': data['y_train'],
  'X_val': data['X_val'],
  'y_val': data['y_val'],
  'X_test': data['X_test'],
  'y_test': data['y_test'],

}
weight_scale = 1e-3 #
dropout = 0.7
model = FullyConnectedNet(hidden_dims, weight_scale=weight_scale, reg = 0, dropout =
solver = Solver(model, whole_data,
                num_epochs=60, batch_size=100,
                update_rule='adam',
                optim_config={'learning_rate': 1e-3,},
                lr_decay = 0.95,
                verbose=True, print_every=200)
solver.train()
# ================================================================ #
# END YOUR CODE HERE
# ================================================================ #
```

```
(Iteration 1 / 29400) loss: 2.298408
(Epoch 0 / 60) train acc: 0.141000; val_acc: 0.151000
(Iteration 201 / 29400) loss: 1.594431
(Iteration 401 / 29400) loss: 1.598612
(Epoch 1 / 60) train acc: 0.463000; val_acc: 0.448000
(Iteration 601 / 29400) loss: 1.723591
(Iteration 801 / 29400) loss: 1.557372
(Epoch 2 / 60) train acc: 0.462000; val_acc: 0.488000
(Iteration 1001 / 29400) loss: 1.598769
(Iteration 1201 / 29400) loss: 1.491483
(Iteration 1401 / 29400) loss: 1.463362
(Epoch 3 / 60) train acc: 0.521000; val_acc: 0.494000
(Iteration 1601 / 29400) loss: 1.509252
(Iteration 1801 / 29400) loss: 1.466622
(Epoch 4 / 60) train acc: 0.545000; val_acc: 0.529000
(Iteration 2001 / 29400) loss: 1.259682
(Iteration 2201 / 29400) loss: 1.257530
(Iteration 2401 / 29400) loss: 1.242700
(Epoch 5 / 60) train acc: 0.578000; val_acc: 0.550000
(Iteration 2601 / 29400) loss: 1.494734
(Iteration 2801 / 29400) loss: 1.319884
(Epoch 6 / 60) train acc: 0.574000; val_acc: 0.548000
(Iteration 3001 / 29400) loss: 1.412498
(Iteration 3201 / 29400) loss: 1.288664
(Iteration 3401 / 29400) loss: 1.226284
(Epoch 7 / 60) train acc: 0.611000; val_acc: 0.556000
(Iteration 3601 / 29400) loss: 1.143371
(Iteration 3801 / 29400) loss: 1.251719
(Epoch 8 / 60) train acc: 0.623000; val_acc: 0.555000
(Iteration 4001 / 29400) loss: 1.249114
(Iteration 4201 / 29400) loss: 1.172773
(Iteration 4401 / 29400) loss: 1.119578
(Epoch 9 / 60) train acc: 0.633000; val_acc: 0.559000
(Iteration 4601 / 29400) loss: 1.229059
(Iteration 4801 / 29400) loss: 1.262048
(Epoch 10 / 60) train acc: 0.636000; val_acc: 0.564000
(Iteration 5001 / 29400) loss: 1.354756
(Iteration 5201 / 29400) loss: 1.420874
(Epoch 11 / 60) train acc: 0.641000; val_acc: 0.569000
(Iteration 5401 / 29400) loss: 1.230693
(Iteration 5601 / 29400) loss: 1.043646
(Iteration 5801 / 29400) loss: 1.024204
(Epoch 12 / 60) train acc: 0.641000; val_acc: 0.581000
(Iteration 6001 / 29400) loss: 1.121003
(Iteration 6201 / 29400) loss: 1.223647
(Epoch 13 / 60) train acc: 0.633000; val_acc: 0.560000
(Iteration 6401 / 29400) loss: 0.978449
(Iteration 6601 / 29400) loss: 1.315928
(Iteration 6801 / 29400) loss: 1.060285
(Epoch 14 / 60) train acc: 0.656000; val_acc: 0.575000
(Iteration 7001 / 29400) loss: 1.254175
(Iteration 7201 / 29400) loss: 1.071293
(Epoch 15 / 60) train acc: 0.703000; val_acc: 0.575000
```

```
(Iteration 7401 / 29400) loss: 1.051179
(Iteration 7601 / 29400) loss: 0.993687
(Iteration 7801 / 29400) loss: 1.239175
(Epoch 16 / 60) train acc: 0.661000; val_acc: 0.580000
(Iteration 8001 / 29400) loss: 1.368725
(Iteration 8201 / 29400) loss: 0.893911
(Epoch 17 / 60) train acc: 0.725000; val_acc: 0.568000
(Iteration 8401 / 29400) loss: 0.965442
(Iteration 8601 / 29400) loss: 1.009901
(Iteration 8801 / 29400) loss: 1.008127
(Epoch 18 / 60) train acc: 0.705000; val_acc: 0.583000
(Iteration 9001 / 29400) loss: 0.983968
(Iteration 9201 / 29400) loss: 1.203774
(Epoch 19 / 60) train acc: 0.744000; val_acc: 0.582000
(Iteration 9401 / 29400) loss: 1.004626
(Iteration 9601 / 29400) loss: 1.013244
(Epoch 20 / 60) train acc: 0.716000; val_acc: 0.588000
(Iteration 9801 / 29400) loss: 1.084729
(Iteration 10001 / 29400) loss: 0.885756
(Iteration 10201 / 29400) loss: 1.114381
(Epoch 21 / 60) train acc: 0.726000; val_acc: 0.592000
(Iteration 10401 / 29400) loss: 1.118271
(Iteration 10601 / 29400) loss: 1.087094
(Epoch 22 / 60) train acc: 0.730000; val_acc: 0.600000
(Iteration 10801 / 29400) loss: 1.047215
(Iteration 11001 / 29400) loss: 1.079156
(Iteration 11201 / 29400) loss: 1.115825
(Epoch 23 / 60) train acc: 0.761000; val_acc: 0.591000
(Iteration 11401 / 29400) loss: 0.987570
(Iteration 11601 / 29400) loss: 1.156316
(Epoch 24 / 60) train acc: 0.748000; val_acc: 0.587000
(Iteration 11801 / 29400) loss: 0.899250

(Iteration 12001 / 29400) loss: 0.930228
(Iteration 12201 / 29400) loss: 0.869270
(Epoch 25 / 60) train acc: 0.748000; val_acc: 0.607000
(Iteration 12401 / 29400) loss: 0.982034
(Iteration 12601 / 29400) loss: 0.998646
(Epoch 26 / 60) train acc: 0.783000; val_acc: 0.591000
(Iteration 12801 / 29400) loss: 0.799769
(Iteration 13001 / 29400) loss: 1.040560
(Iteration 13201 / 29400) loss: 0.965084
(Epoch 27 / 60) train acc: 0.779000; val_acc: 0.589000
(Iteration 13401 / 29400) loss: 0.842854
(Iteration 13601 / 29400) loss: 1.041773
(Epoch 28 / 60) train acc: 0.741000; val_acc: 0.580000
(Iteration 13801 / 29400) loss: 0.942940
(Iteration 14001 / 29400) loss: 1.035560
(Iteration 14201 / 29400) loss: 0.768751
(Epoch 29 / 60) train acc: 0.790000; val_acc: 0.589000
(Iteration 14401 / 29400) loss: 0.847572
(Iteration 14601 / 29400) loss: 0.873548
(Epoch 30 / 60) train acc: 0.798000; val_acc: 0.582000
(Iteration 14801 / 29400) loss: 0.948969
```

```
(Iteration 15001 / 29400) loss: 0.902381
(Epoch 31 / 60) train acc: 0.793000; val_acc: 0.592000
(Iteration 15201 / 29400) loss: 0.772018
(Iteration 15401 / 29400) loss: 0.882011
(Iteration 15601 / 29400) loss: 0.807604
(Epoch 32 / 60) train acc: 0.786000; val_acc: 0.591000
(Iteration 15801 / 29400) loss: 0.765125
(Iteration 16001 / 29400) loss: 0.786065
(Epoch 33 / 60) train acc: 0.823000; val_acc: 0.582000
(Iteration 16201 / 29400) loss: 0.905458
(Iteration 16401 / 29400) loss: 0.811378
(Iteration 16601 / 29400) loss: 0.824763
(Epoch 34 / 60) train acc: 0.792000; val_acc: 0.601000
(Iteration 16801 / 29400) loss: 0.920568
(Iteration 17001 / 29400) loss: 0.926238
(Epoch 35 / 60) train acc: 0.794000; val_acc: 0.592000
(Iteration 17201 / 29400) loss: 0.840497
(Iteration 17401 / 29400) loss: 0.818339
(Iteration 17601 / 29400) loss: 1.023683
(Epoch 36 / 60) train acc: 0.793000; val_acc: 0.601000
(Iteration 17801 / 29400) loss: 1.073276
(Iteration 18001 / 29400) loss: 0.973836
(Epoch 37 / 60) train acc: 0.811000; val_acc: 0.610000
(Iteration 18201 / 29400) loss: 0.973277
(Iteration 18401 / 29400) loss: 0.957541
(Iteration 18601 / 29400) loss: 0.991387
(Epoch 38 / 60) train acc: 0.827000; val_acc: 0.601000
(Iteration 18801 / 29400) loss: 0.745107
(Iteration 19001 / 29400) loss: 0.799122
(Epoch 39 / 60) train acc: 0.811000; val_acc: 0.598000
(Iteration 19201 / 29400) loss: 0.942136
(Iteration 19401 / 29400) loss: 0.800624

(Epoch 40 / 60) train acc: 0.783000; val_acc: 0.604000
(Iteration 19601 / 29400) loss: 0.940429
(Iteration 19801 / 29400) loss: 0.904573
(Iteration 20001 / 29400) loss: 0.993399
(Epoch 41 / 60) train acc: 0.816000; val_acc: 0.591000
(Iteration 20201 / 29400) loss: 0.899653
(Iteration 20401 / 29400) loss: 0.916943
(Epoch 42 / 60) train acc: 0.806000; val_acc: 0.596000
(Iteration 20601 / 29400) loss: 0.758303
(Iteration 20801 / 29400) loss: 0.778680
(Iteration 21001 / 29400) loss: 0.947486
(Epoch 43 / 60) train acc: 0.811000; val_acc: 0.592000
(Iteration 21201 / 29400) loss: 0.712344
(Iteration 21401 / 29400) loss: 0.806183
(Epoch 44 / 60) train acc: 0.826000; val_acc: 0.605000
(Iteration 21601 / 29400) loss: 0.783382
(Iteration 21801 / 29400) loss: 0.891782
(Iteration 22001 / 29400) loss: 0.834167
(Epoch 45 / 60) train acc: 0.819000; val_acc: 0.606000
(Iteration 22201 / 29400) loss: 0.797323
(Iteration 22401 / 29400) loss: 0.844305
```

```
(Epoch 46 / 60) train acc: 0.854000; val_acc: 0.614000
(Iteration 22601 / 29400) loss: 0.890112
(Iteration 22801 / 29400) loss: 0.944924
(Iteration 23001 / 29400) loss: 0.606655
(Epoch 47 / 60) train acc: 0.836000; val_acc: 0.606000
(Iteration 23201 / 29400) loss: 0.640090
(Iteration 23401 / 29400) loss: 0.970375
(Epoch 48 / 60) train acc: 0.830000; val_acc: 0.594000
(Iteration 23601 / 29400) loss: 0.662647
(Iteration 23801 / 29400) loss: 0.762254
(Iteration 24001 / 29400) loss: 0.780637
(Epoch 49 / 60) train acc: 0.842000; val_acc: 0.611000
(Iteration 24201 / 29400) loss: 0.901651
(Iteration 24401 / 29400) loss: 0.732053
(Epoch 50 / 60) train acc: 0.835000; val_acc: 0.601000
(Iteration 24601 / 29400) loss: 0.806242
(Iteration 24801 / 29400) loss: 1.107829
(Epoch 51 / 60) train acc: 0.844000; val_acc: 0.602000
(Iteration 25001 / 29400) loss: 0.814229
(Iteration 25201 / 29400) loss: 0.809749
(Iteration 25401 / 29400) loss: 0.762215
(Epoch 52 / 60) train acc: 0.852000; val_acc: 0.601000
(Iteration 25601 / 29400) loss: 0.823286
(Iteration 25801 / 29400) loss: 0.756253
(Epoch 53 / 60) train acc: 0.838000; val_acc: 0.605000

(Iteration 26001 / 29400) loss: 1.023917
(Iteration 26201 / 29400) loss: 0.755779
(Iteration 26401 / 29400) loss: 0.876049
(Epoch 54 / 60) train acc: 0.833000; val_acc: 0.598000
(Iteration 26601 / 29400) loss: 0.739671
(Iteration 26801 / 29400) loss: 0.771005
(Epoch 55 / 60) train acc: 0.838000; val_acc: 0.600000
(Iteration 27001 / 29400) loss: 0.625780
(Iteration 27201 / 29400) loss: 0.734696
(Iteration 27401 / 29400) loss: 0.845899
(Epoch 56 / 60) train acc: 0.869000; val_acc: 0.603000
(Iteration 27601 / 29400) loss: 0.761696
(Iteration 27801 / 29400) loss: 0.638063
(Epoch 57 / 60) train acc: 0.851000; val_acc: 0.607000
(Iteration 28001 / 29400) loss: 0.811818
(Iteration 28201 / 29400) loss: 0.958917
(Iteration 28401 / 29400) loss: 1.009080
(Epoch 58 / 60) train acc: 0.869000; val_acc: 0.603000
(Iteration 28601 / 29400) loss: 0.863148
(Iteration 28801 / 29400) loss: 0.497598
(Epoch 59 / 60) train acc: 0.850000; val_acc: 0.604000
(Iteration 29001 / 29400) loss: 0.876637
(Iteration 29201 / 29400) loss: 0.776647
(Epoch 60 / 60) train acc: 0.842000; val_acc: 0.612000
```

In [10]:

```
y_val_pred = np.argmax(model.loss(data['X_val']), axis = 1)
y_test_pred = np.argmax(model.loss(data['X_test']), axis = 1)
print('Validation accuracy is:', np.mean(y_val_pred == data['y_val']))
print('Testing accuracy is:', np.mean(y_test_pred == data['y_test']))
```

```
Validation accuracy is: 0.604
Testing accuracy is: 0.612
```

In [ ]:

```

```

In [ ]:

```

```